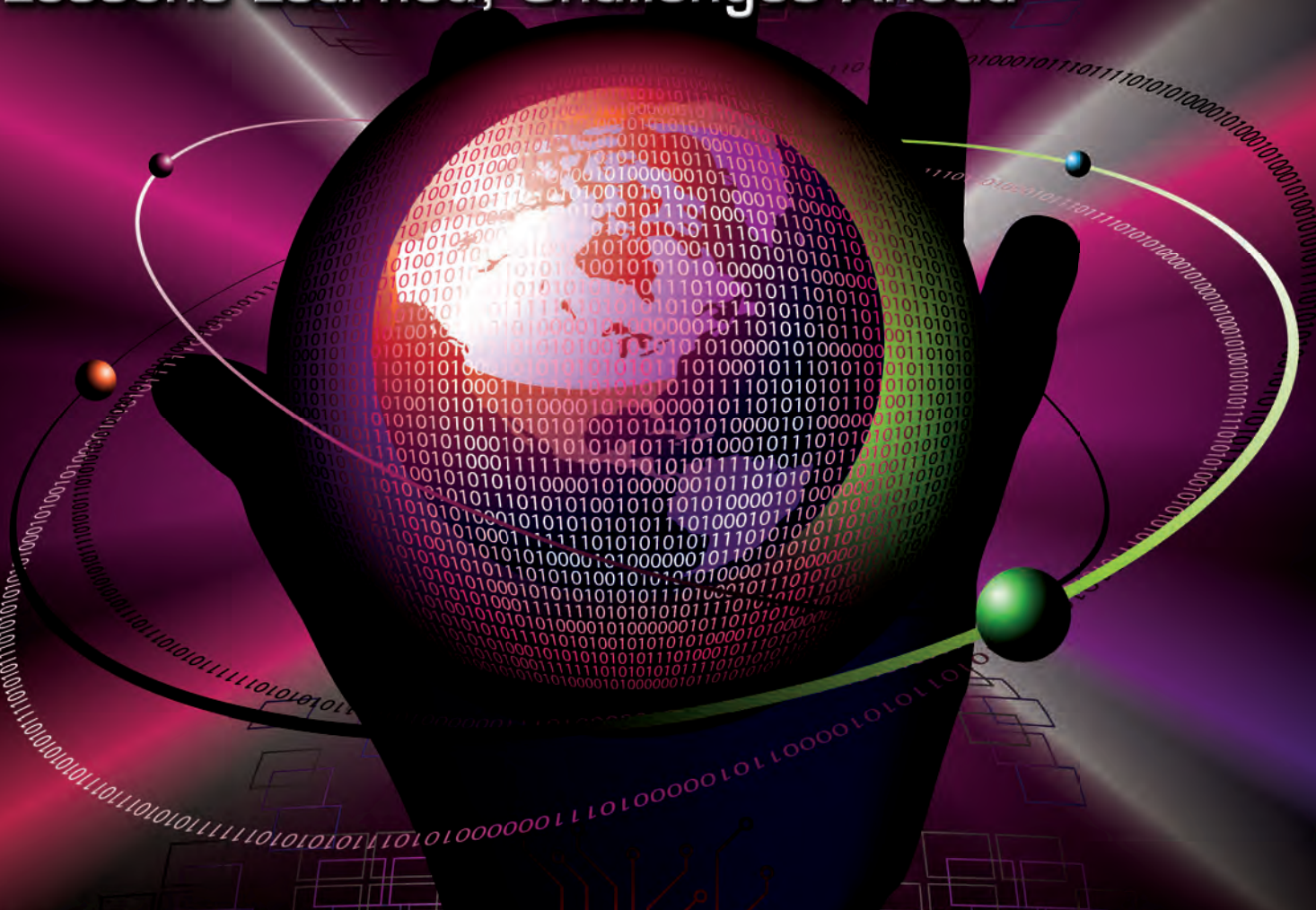
February 2011 Vol.14 No.1

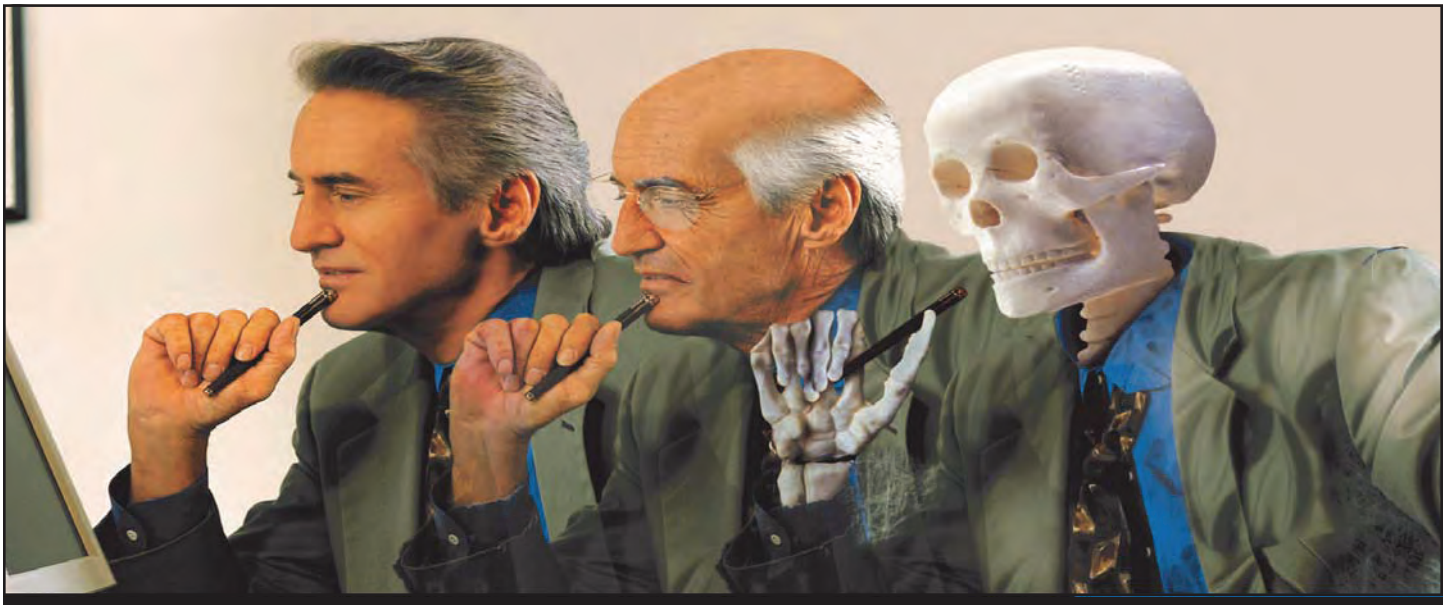
SoftwareTech NEWS
www.softwaretechnews.com

DoD and Open Source Software

Lessons Learned, Challenges Ahead



DACS
http://iac.dtic.mil/dacs
Unclassified and Unlimited Distribution



How long can you wait for CMMI® Compliance?

Manage your projects in guaranteed compliance with the CMMI — Now!

Don't waste valuable time and resources developing CMMI-compliant processes from scratch when there is a proven approach that guarantees success. With **processMax®,** you begin operating in compliance *immediately:* no process development is required!

processMax is a complete project management system, integrated with Microsoft Project, and is guaranteed by **pragma** SYSTEMS to be compliant with CMMI-DEV.

With **processMax**, managers and their teams efficiently collaborate with step-by-step procedures, integrated document management, risk management, automated workflow, and automated measurement and reporting. **processMax** increases productivity, reduces defects, and manages risk.

Now available as a hosted service for both our subscription and perpetual licenses, **processMax** is more affordable than ever. We manage the server, installation, updates, and upgrades.

More than 70 organizations have passed Level 2 and Level 3 appraisals with **processMax**, at a fraction of the time and expense required by traditional methods.

Please contact us to learn how processMax, can help you achieve your compliance goals.

processMax
THE PROCESS SOLUTION

pragma SYSTEMS CORPORATION

www.pragmasystems.com
703-796-0010
info@pragmasystems.com

GSA Schedule Contract NO. GS-35F-0559S. processMax is a registered trademark of **pragma** SYSTEMS CORPORATION.

Although processMax makes use of portions of "CMMI for Development, Version 1.2," CMU/SEI-2006-TR-008, copyright 2006 by Carnegie Mellon University, neither the Software Engineering Institute nor Carnegie Mellon University have reviewed or endorsed this product.

This is a paid advertisement.

TECH VIEWS

By John Dingman, Editor

It has been three and a half years since Software Tech News addressed the topic of Open Source Software (OSS). In that time, as *Kane McLean* writes in his article, "**Military Open Source Community Growing**", its use in the Department of Defense (DoD) has grown significantly, widely adopted and implemented in a variety of systems.

While the rate of change in technology grows exponentially, DoD needs to continue to develop new capabilities ever faster. Gone forever are the days of a single contractor developing a system from scratch, uniquely matched to the required application. The need for commonality across platforms, interoperability between networks and shared functionality across organizations drive reusing what exists rather than reinventing the same basic constructs. Matthew Kennedy discusses the factors one should consider in his article, "**Evaluating Open Source Software**".

As agencies look for ways to cut development costs while reducing development time the availability of open source components becomes more than just an attractive alternative. But simply downloading a publicly available component and including it in a project can lead to significant repercussions.

Dr. David A. Wheeler points out in his article, "**Open Source Software (OSS) is Commercial**", it is in fact a commercial product, often developed to make a profit. *John Scott, Dr. Wheeler, Mark Lucas and J.C. Herz* discuss licensing and intellectual property issues in "**Running Open Technology Development Projects**". *Lawrence Rosen* explains many of the issues concerning copyrights and patents in his article, "**Implementing Open Standards in Open Source**".

Failure to recognize these technical and legal implications can impact the long term usability of a developed product. In the last three years of OSS use in the DoD there have been many lessons learned, one of them being that there are still challenges ahead.

Author Contact Information

Email: news-editor@thedacs.com

The DACS OSS topic page contains information relative to the development, use, licensing and promotion of open source software including operating systems, browsers and applications: https://www.thedacs.com/databases/url/key/4878

DILBERT

Software is a Renewable Military Resource

By John Scott, Dr. David A. Wheeler, Mark Lucas, and J.C. Herz

"The United States cannot retreat behind a Maginot Line of firewalls or it will risk being overrun. Cyberwarfare is like maneuver warfare, in that speed and agility matter most."

– William J. Lynn III. [Lynn2010]

Software is the fabric that enables planning, weapons and logistics systems to function: it might be the only infinitely renewable military resource.

In particular, DoD must have a software environment that is easily adaptable to changing mission needs; this software must also evolve at lower cost and be delivered rapidly so it can be used when it is needed. This technological evolution entails a parallel evolution in acquisitions methodologies and corporate attitude to facilitate discovery, re-use, and modification of software across the DoD and U.S. Government. A new way is needed to develop, deploy and update software-intensive systems that will match the tempo and ever-changing mission demands of military operations.

Software code has become central to how the war-fighter conducts missions. If this shift is to be a strength, rather than an Achilles' heel, DoD must pursue an active strategy to manage its software portfolio and foster a culture of open interfaces, modularity and reuse [Scott2010]. Moving forward, the government needs to define a modern software intellectual property regime to broaden the defense industrial base by enabling industry-wide access to defense knowledge, thereby increasing competition and eventually lowering the cost of innovation. Over time, the military would evolve common software architectures and industry-wide baselines to increase the adaptability, agility and - most important - capacity to meet new dynamic threats.

Military Open Technology Development (OTD) Strategy

"In a real world of limited resources and skills, individuals and groups form, dissolve and reform their cooperative or competitive postures in a continuous struggle to remove or overcome physical and social environmental obstacles. Technological agility should be a metric."

– Col John Boyd (USAF) [Boyd1976]

Open Technology Development (OTD) has become an approach to military software/system development in which developers (outside government and military) collaboratively develop and manage software or a system in a decentralized fashion. OTD depends on open standards and interfaces, open source software and designs, collaborative and distributed online tools, and technological agility. [OTD2006]

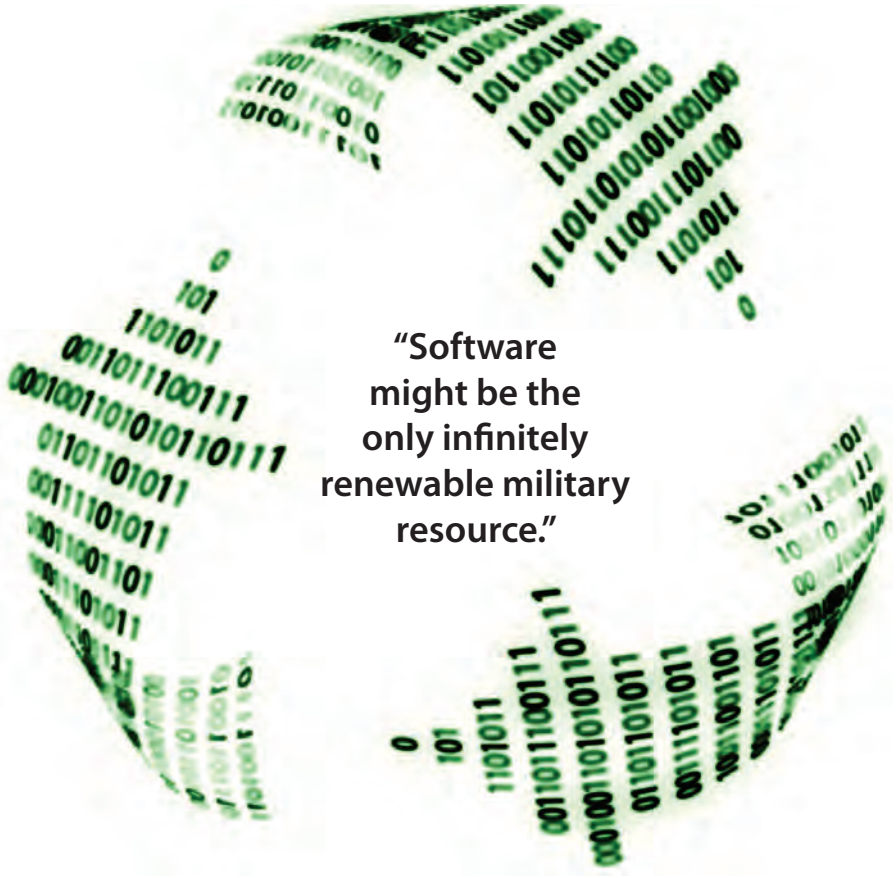
These practices are proven and in use in the commercial world. Open standards and interfaces allow systems and services to evolve in a shifting marketplace. Using, improving, and developing open source software minimizes redundant software engineering and enables agile development of systems. Collaborative and distributed online tools are now widely used for software development. The private sector also often strives to avoid being locked into a single vendor or technology and instead tries to keep its technological options open (e.g., by adhering to open standards). Previous studies have documented that open source software is currently used in many of DoD's critical applications and is now an inseparable part of military infrastructure [MITRE2003] [OTD2006].

OTD methodologies rely on the ability of a software community of interest to access software code or application interfaces across the enterprise. This access to source code, design documents and to other developers and end-users enables decentralized development of capabilities that leverage existing software assets. OTD methodologies have been used for open source development, open standards architectures, and the most recent generation of web-based collaborative technologies. The most successful implementations come from direct interaction with the end-user community. The open source software development model is successful because communities of interest involve both developers and users.

OTD includes open source initiatives but is not limited to open source software (OSS) development and licensing regimes, which enforce redistribution of code. It is important, in the context of this report and resulting policy discussions, to distinguish between OSS and OTD, since the latter may include code whose distribution may be limited to DoD, and indeed may only be accessible on classified networks. Nor does the promotion of OTD within DoD impinge on the legal status of software developed by with private sector money by commercial vendors.

Some key benefits of OTD are listed below and in the following articles in this issues of DACS:

- Increased Agility/Flexibility: Because the government has unrestricted access and rights to the source code it has paid to develop, and can therefore make that code discoverable and accessible to program managers and contractors alike, it is possible to find an "80% solution" and modify it for a new mission. Likewise, pre-existing government-funded components from different programs can be assembled without having to hack through a thicket of intellectual property rights which require lawyers to negotiate. Instead of having to start from scratch every time it wants to develop a capability, the government can find what works and draw from a broad base of developers and contractors who can rapidly assemble and modify existing systems and components.
- Faster delivery: because developers only need to focus on changes to, and integration of, existing software capabilities, instead of having to redevelop entire systems, they can cut the time to delivery for new capabilities. Even when a module or component is developed from scratch to replace an outdated one, it benefits from open interfaces and standards in the systems with which it interacts. With "goes intas and goes outtas" in hand, development and deployment time can be cut.
- Increased Innovation: Because they have access to the source code for existing capabilities, developers and contractors can focus their time and effort on innovation, i.e. writing the code that takes existing capabilities to a new level, or synthesizes components into a whole that's greater than the sum of its parts. This is particularly important because of a projected shortfall in the number of U.S. citizens with engineering and computer science degrees who will be clearable to work on military projects in the coming decades [National Academies 2008]. As a greater proportion of software engineering degrees are held by foreign nationals, and U.S. programmers are lured by innovative and lucrative work in the private sector, the military will face a long-term shortage of software engineers to work on military-specific systems. The Defense Department therefore must focus on the long-term challenge of getting more innovation out of a restricted talent pool. It will be important to leverage that human capital by having engineers focus on the 10% of source code that actively improves a system, vs. the 90% that's there just to allow a system to plug into existing networks and perform pre-existing functions.
- Information Assurance & Security: One of the biggest values of open source development is enabling wider community access to software source. In this manner



all bugs become shallow and more easily found. Wider access to software source code also is key for forming and maintaining a software security posture from being able to review software source code to seeing what is actually present within that software.

- Lower cost: The first cost to fall by the wayside with OTD is the monopoly rent the government pays to contractors who have built a wall of exclusivity around capabilities they've been paid by the government to develop. They may have internally developed source code (IRAD – internal research and development) that's valuable, but in an OTD system that code has been modularized so the government can make a rational decision about whether they want to re-license it for a new project or pay to develop a replacement. The entire value of the government's investment hasn't been voided by the mingling of IRAD into a government funded system as a means of ensuring lock-in to a particular vendor. With unlimited rights and access to government-funded source code, the government can draw on a broader pool of competitive proposals for software updates and new capabilities that leverage current systems. The elimination of monopoly rent, combined with greater competition, will drive down costs and improve the quality of resulting deliverables, because any contractor who works on a system knows that they can be replaced by a competitor who has full access to the source code and documentation of an OTD system.

Off-the-shelf (OTS) Software Development Approaches, including Open Government OTS (OGOTS) and Open Source Software (OSS)

Military programs must adapt and move its software and technologies away from passively managed and closed GOTS (Government Off-the-Shelf) programs toward Open Government Off-the-Shelf (OGOTS) and ultimately toward Open Source Software (OSS) for maximum flexibility and agility.

Open Technology Development involves community development among government users, and thus includes both OSS and OGOTS. An OTD strategy allows organizations to develop and maintain software in a collaborative way. To maximize collaboration, software should be developed to use off-the-shelf (OTS) components and to be itself OTS to the maximum practical extent.

Off-the-shelf (OTS) software is simply software that is ready-made and available for use. The rationale for developing OTS software is to create software that can be used for multiple purposes, instead of using custom-built software for a single purpose and use. OTS software has the potential to save time, save money, increase quality, and increase innovation through resource pooling. Even when a custom system is needed, building it from many OTS components has many advantages.

There are many different ways that off-the-shelf (OTS) software can be maintained. Some OTS may be retained and maintained inside the U.S. government (e.g., because it is classified or export controlled); such software is termed government OTS (GOTS). Off-the-shelf items that are commercial items (e.g., by being sold, licensed, or leased to the public for non-governmental use) are commercial OTS (COTS). Note that by law and regulation, software licensed to the public and used for at least one non-government purpose is COTS software, even if it is maintained by the government. Figure 1 illustrates these different kinds of OTS maintenance approaches.There are two kinds of commercial OTS (COTS) software: Open Source Software (OSS) and proprietary software. In either case they may be maintained by a single maintainer or by a community. In community maintenance there is often a single organization who determines if proposals should be accepted, but the key here is that the work tends to be distributed among those affected.

Today, where there is GOTS software at all, it tends to be developed and maintained by a single maintainer. This tends to reduce GOTS' applicability. Many government programs might potentially use a GOTS component if certain changes were made, but cannot make the changes to the GOTS component directly, and even if they did, there is no structure by which those changes could be merged back into the main GOTS product for all to use. In contrast, most OSS projects are maintained by communities, where different organizations actively work together to develop software that is useful to them all. Single-maintainer OSS project exist, but they are less common.

An Open GOTS (OGOTS) project is a GOTS project which uses multiple-organization collaborative development approaches to develop and maintain software, in a manner similar to OSS. Such a project within the DoD is sometimes termed "DoD community source software." One goal of this paper is to increase the number of GOTS projects that are OGOTS projects. A project may become OGOTS instead of OSS because its leaders want the innovation, speed of development, and lowered cost that can come from co-development by many parties, yet:

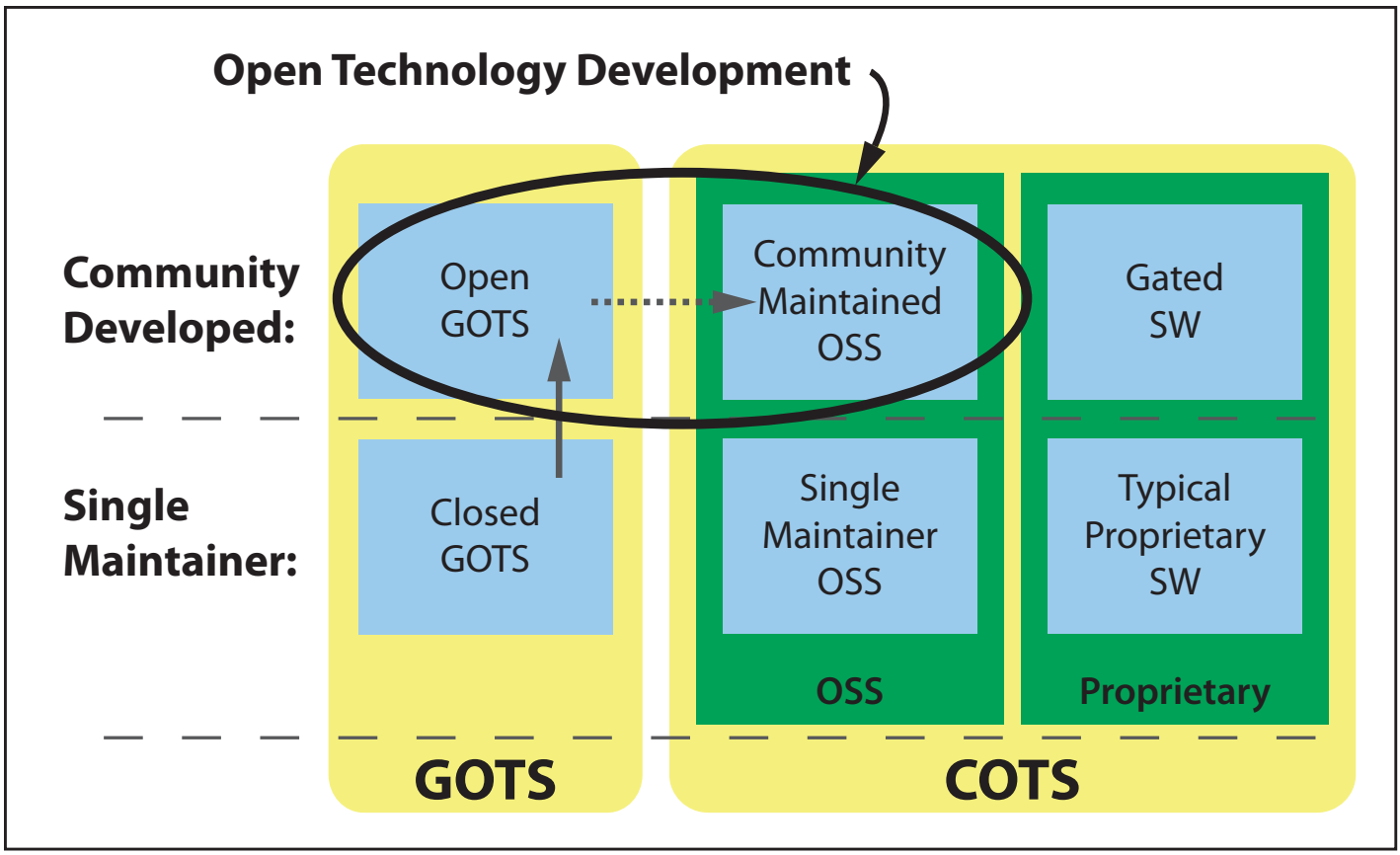


Figure 1: Off-the-Shelf (OTS) Maintenance Strategies

1. The government lacks the intellectual rights to make it more open (e.g., the government may have government-purpose rights (GPR) and not unlimited rights), and/or

2. The government wishes to maintain a national security advantage by not making that software available to potential adversaries (typically such software will be classified and/or export controlled).

In addition, GOTS projects should determine when they should become COTS (e.g., as community-supported OSS projects). In particular, GOTS projects should seriously consider switching to OSS maintenance after a system has been deployed. There are various reasons why the government should keep certain software in-house, e.g., because sole possession of the software gives the U.S. a distinct advantage over its adversaries. However, technological advantage is usually fleeting. Often there is a commercially-developed item available to the public that begins to perform similar functions. As it matures, other organizations begin using this non-GOTS solution, potentially rendering the GOTS solution obsolete. Such cases often impose difficult decisions, as the government must determine if it will pay the heavy asymmetrical cost to switch, or if it will continue "as usual" with its now-obsolete GOTS systems (with high annual costs and limitations that may risk lives or missions). This means that there is considerable risk to the government if it tries to privately hold GOTS software within the government for too long.

As Defense Secretary Robert Gates said "The gusher has been turned off and will stay off for a good period of time." DoD needs a more efficient software development ecosystem – more innovation at lower cost - and OTD squeezes financial waste out of the equation by reducing lock-in and increasing competition.

References

[Boyd1976] Boyd, Col. John, *Destruction and Creation - John Boyd - Winning and Losing*, Sept 3, 1976

[Lynn2010] Lynn, William J. III. September 2010. "Defending a New Domain: The Pentagon's Cyberstrategy". Foreign Affairs.

[MITRE2003] MITRE Corporation. January 2, 2003. Use of Free and Open-Source Software (FOSS) in the U.S. Department of Defense. http://cio-nii.defense.gov/sites/oss/2003Survey/dodfoss_pdf.pdf

[National Academies 2008] Rising Above the Gathering Storm: Energizing and Employing America for a Brighter Economic Future. Report of the Committee on Science, Engineering, and Public Policy. National Academies Press, 2008

[OTD2006] J.C. Herz, Mark Lucas, and John Scott. April 2006. *Open Technology Development Roadmap Plan*. http://www.acq.osd.mil/jctd/articles/OTDRoadmapFinal.pdf.

[Scott2010] Scott, John. 2010. Pentagon is Loosing the Softwar(e). Denfense News June 21, 2010. http://www.defensenews.com/story.php?i=4677662

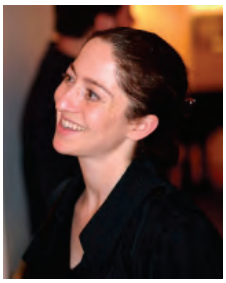
About the Authors

John Scott is a Senior Systems Engineer and Open Technology Lead forRadiantBlue Technologies, Inc. with expertise in engineered systems and bridging the gap between decision-makers, scientists, and engineers to develop policies for acquiring and deploying new technologies in the Department of Defense and US Government.

John helped draft the U.S. Department of Defense policy for the use of open source software as well as founded (now co-chairman) Open Source for America, an advocacy group for use of open source software in government and the Military Open Source Software workign group (http://mil-oss.org/). He is also a member of the Council on Foreign Relations. He holds a BS in Mechanical Engineering from Lehigh University, an MS in Systems Engineering from Virginia Tech. and serves as the chairman of the National Defense Industrial Association's Command, Control, Communications and Computers (C4) division. He writes about defense software and acquisitions related issues, most recently at Defense News entitled "Pentagon Is Losing the Softwar(e)." jms3rd@gmail.com, jscott@radiantblue.com.

Dr. David A. Wheeler works at the Institute for Defense Analyses (IDA). Dr. Wheeler is a well-known expert in developing secure software and open source software. His works on security include the book "Secure Programming for Linux and Unix HOWTO," his 'developerWorks' column "Secure Programmer," and the article "Countering Trusting Trust through Diverse Double-Compiling (DDC)." He is a co-author of the NDIA document "Engineering for System Assurance," and the author of "Why Open Source Software? Look at the Numbers!", a collection of quantitative studies where he argued that considering OSS is justified. This article has been cited in over 80 scholarly works and in government reports such as the 2004 report of the California Performance Review.

Dr. Wheeler has previously written on the issue of OSS being commercial. This is a critical issue in U.S. federal government acquisitions, because the U.S. government has laws and policies that prefer the acquisition of commercial items.



J.C. Herz is a technologist with a background in biological systems and computer game design. Her specialty is massively multiplayer systems that leverage social network effects, whether on the web, mobile devices, or more exotic high-end or grubby low-end hardware. Her defense projects range from aerospace systems to a computer-game derived interface for next generation unmanned air systems. J.C. was a member of the National Research Council's committee on IT and Creative Practice, and is a founding member of the IEEE Task Force on Game Technologies.

Mr. Mark Lucas has pioneered efforts in Open Source Software Development in remote sensing, image processing, and geographical information systems. Mark established remotesensing.org and has led several government funded studies and development efforts since 1996. These efforts include the Open Source Software Image Map (OSSIM) projects for the NRO, the Open Source Prototype Research, and the Open Source Extraordinary Program projects for NGA. He is currently leading the Open Technology Development effort within DoD AS&C in collaboration with NII, and the Business Transformation Agency.

Mark has a BS in Electrical Engineering and Computer Science from the University of Arizona and a MS in Computer Science from West Coast University. He was commissioned in the Air Force and assigned to the Secretary of the Air Force Special Projects organization. He has experience as both a government and contractor program manager through a number of classified programs. He is on the Board of Directors of the Open Source Geo-spatial Foundation, the Open Source Software Institute, and the National Center for Open Source Policy and Research. Mark is currently a principal scientist at RadiantBlue Technologies, Inc.

Work sponsored by the Assistant Secretary of Defense (Networks & Information Integration) (NII) / DoD Chief Information Officer (CIO) and the Under Secretary of Defense for Acquisition, Technology, and Logistics (AT&L)

Military Open Source Community Growing

By Kane McLean, BRTRC Technology Research Corporation

Open source software (OSS) usage is growing across the Department of Defense (DoD), not surprisingly so is the community of open source developers and integrators. In industry and the technology community at large many consider open source to be somewhat of a social movement centered around the free exchange of technological ideas; however across the DoD, where pragmatism so common in the Information Technology (IT) landscape, OSS is often simply the best solution to the military's technology challenges.

In 2009 the DoD issued the policy memo Clarifying Guidance Regarding Open Source Software (mil-oss.org/resources/us-dod_policy-memo_clarifying-guidance-regarding-oss_16oct2009.pdf) which defines "open source software" as "software for which the human-readable source code is available for use, study, reuse, modification, enhancement, and redistribution by the users of that software. In other words, OSS is software for which the source code is 'open'." Although minor issues such as some Security Technical Implementation Guides (STIG) compliance remain to be worked out, this memo clears the last substantive policy obstacle for OSS use within the DoD and its services.

Open source projects in the private sector naturally attract contributors and organized community involvement, it is only natural that the same is happening now that the DoD has embraced OSS. One of the first groups to gain momentum is Military Open Source (Mil-OSS). Each year, the group hosts a convention called a Working Group where members from all over the United States come together to learn, share and discuss OSS projects, upcoming policy changes and how to understand and support the military in its adoption of OSS.

WG2: The 2010 Military Open Source Conference

In August of 2010, Mil-OSS hosted its second annual Working Group (WG2). The speakers and topics for the 2010 conference reflected not only applying OSS to current DoD IT challenges, but the hot IT issues DoD faces at large such as Cyber Security, DoD Social Platforms, Cloud Computing, CMS Platforms, and more.

The WG2 speakers included people from government, military and industry. Lt. Gen. Robert J. Elder, Jr. (Ret.) delivered a keynote which discussed the challenges of modern loose working structures operating within and against the hierarchical environment of the DoD. Lt. Col. Nate Allen's keynote discussed the of adoption of gated social networks by the U.S. Army for professional development through the Company Command and Platoon Leader forums as well as the ORION project which brings social workflows to the Army's senior leaders. H.D. Moore discussed Open Source Cyber Weaponry in his keynote address.

The roster of 45 speakers included government/military employees and contractors from organizations including the DoD, Army, Navy, Marine Corps, Air Force, Red Hat, Georgia Tech Research Institute, BRTRC, Boeing, Lockheed Martin, SecureForce, Geocent, IDA, Acquia, QinetiQ, Open Information Security Foundation, DHS, DISA, HHS, among others.

Daniel Risacher, who helped craft the recent OSS guidance from the DoD CIO, discussed the development and impact of the new policy memo and held an open Question and Answer session for the community about it.

Anyone who also attended WG1 (the 2009 Mil-OSS Working Group), could not have escaped noticing how much traction the Open Source community has gained in the DoD within the last twelve months. With the new DoD OSS guidance, the overall community tone has shifted from "can we?" to "the game is afoot!"

Mil-OSS is an active grass-roots organization that connects and empowers an active community of both civilian and military developers using, improving, and releasing Open Source Software and hardware across the United States Department of Defense. Most members of Mil-OSS work with the DoD either directly or as contractors, and see their work in open source DoD projects as a patriotic support for our country's warfighters. OSS allows the DoD to improve software security, control development costs and increase innovation—all of which benefit the Warfighter.

About Mil-OSS

The idea for the organization goes back to 2003 when Josh Davis, one of the founders, began a dialogue with James Neushul about open-sourcing a project they were working on. Next Josh met Heather Burke with whom he shared the

"Mil-OSS is an active grass-roots organization that connects and empowers an active community of both civilian and military developers using, improving, and releasing Open Source Software (OSS) and hardware across the United States Department of Defense (DoD)."

idea of pulling together a military based open source software conference. Heather introduced him to John Scott, the other founder of Mil-OSS. Over the next several months key individuals joined the conversation including Mike Howard, Gunnar Hellekson, Kit Plummer and Kane McLean.

By 2008, the organization had taken shape and the first annual Working Group (WG1) was being planned for the summer of the following year. Needing a quick online rally-point for the new community, the Mil-OSS Google Group was started along with other social media outlets to support the organization.

About the same time, the Mil-OSS decided to team up with Open Source For America (OSFA). Mil-OSS decided that operating as a working group under OSFA's umbrella would benefit both communities more than working separately.

Looking Forward

With the DoD policy favoring open source practices and software as benefit to national defense, it's no wonder that Mil-OSS community is growing daily. The annual Working Group will be held each year, and local MeetUps and BarCamps are being established in cities across the country so the community can continue to work together throughout the year. It's a great time to be involved in Open Source projects in the Military; we're all looking forward to the innovation that will come from it.

About the Author

Kane McLean is part of the Strategy & Communications Group at BRTRC Technology Research Corporation and currently works supporting the Under Secretary of the Army at the United States Army Office of Business Transformation.

An Open Source advocate, evangelist and strategist, Kane works to find the right solutions to clients' challenges—more often than not Open Source solutions provide the best-fit and prove to be the most flexible long-term solutions. In addition to creating the strong solution strategies, Kane works to tout the business case for Open Source adoption.

Prior to joining BRTRC, Kane led a company that helped corporations and organizations develop their online presence, enhance their strategic communication and improve their collaboration capabilities by adopting open source web-based technologies. Kane is a former entrepreneur having founded two companies, the last of which he sold in 2008 before becoming a defense contractor.

Kane also serves on the Steering Committees of both OSFA and Mil-OSS.

Evaluating Open Source Software

By Matthew Kennedy

There is an overwhelming amount of open source software (OSS) available today that can be used throughout the software development life cycle. Nowadays, it is generally not a matter of whether one should use OSS, but rather, where one should use it. If one were to start a new software development project, he would probably begin by looking for various types of software to aid in development, such as an integrated development environment, version control system, and a bug tracking tool, to name a few. If he looked exclusively for OSS, he could use Eclipse for the integrated development environment, Subversion for the version control system, and Bugzilla for the bug tracking tool. Those products are available for download and are open source. Looking outside the development environment, one's deployed system may require a database. A person could use a proprietary database such as Microsoft® Access, Microsoft® SQL Server, Oracle®, or an open source option such as MySql. When looking to fill a technological need, OSS may be a viable option.

In July 2008, the U.S. Air Force Office of Advanced Systems and Concepts funded Georgia Tech Research Institute to create and release an open source version of FalconView. Used by the Department of Defense since the 1990s, FalconView is a comprehensive mapping tool that supports various mapping formats and includes ample map analysis tools. With both government and private applications moving to open source development, the proper evaluation of OSS throughout the program is imperative to making informed decisions that could affect the life cycle of the project. What are some of the factors that must be considered when choosing whether to use OSS?

What is OSS?

According to a DoD chief information officer memorandum of 2009, "Clarifying Guidance Regarding Open Source Software," OSS is "Software for which the human-readable source code is available for use, study, reuse, modification, enhancement, and redistribution by the users of that software."

That definition of OSS could apply to various terms used throughout federal and DoD guidance and directives. The Federal Acquisition Regulation/Defense Federal Acquisition Regulation Supplement defines commercial computer software

as "Any item, other than real property, that is of a type customarily used by the general public or by non-governmental entities for purposes other than governmental purposes, and (i) Has been sold, leased, or licensed to the general public; or (ii) Has been offered for sale, lease, or license to the general public."

Chapter four of the Defense Acquisition Guidebook defines non-developmental software as "Any software that is not legacy software for the program, or is not developed as part of the effort being accomplished by the developer team. This includes COTS software, government furnished software, open source software, and software being reused from another program."

These definitions show that although OSS is not explicitly defined in DoD guidance and directives, the terms already in place clearly fit. Some OSS projects are as big as, if not bigger than, their proprietary counterparts. According to its website, MySQL, an open source database application, has had more than 100 million copies of its software downloaded or distributed throughout its history and is currently on release 5.5.

Open source software is generally thought to be free as in it has no costs. Though that is true in most cases, generally the term "free" is used in reference to the liberty of interested parties to freely distribute the source code. That is an important aspect to keep in mind when considering the use of OSS—there may be a cost.

Like proprietary software, OSS comes with licenses such as the GNU or Apache license. This article does not cover the licensing associated with OSS; however, it is important that the proper legal representative reviews the license prior to making the final decision. This assures that the manner in which interested parties intend to use the OSS is in accordance with the license.

Is OSS an Open System?

There is no direct correlation between an open system and OSS. Open source specifies that the human-readable source code of the application is available. In contrast, an open system, as defined by the Open Systems Joint Task Force, is specified as "A system that employs modular design, uses widely supported and consensus based standards for its key interfaces, and has been subjected to successful validation and verification tests to ensure the openness of its key interfaces."

The question as to whether OSS meets the definition of an open system must be addressed per DoD Directive 5000.01: "A modular, open-systems approach shall be employed, where feasible." Because there are generally many contributors to open source projects, they tend to have a modular design; however, this is not always the case. Open Office has 450,000 members that have joined the project, so enforcing a modular design is paramount for continued success. Without a modular design, it would be extremely difficult to modify the source code of such a large application with so many contributors.

Another part of the open system definition is using consensus- based standards for key interfaces; this is also referred to as using open standards. Open standards play a critical role in our systems with modifiability, maintainability, and increased competition. Open standards have no direct correlation to OSS. Though most OSS projects use open standards, it is not required. Each OSS project must be assessed individually to determine if it is, indeed, an open system.

Are the Releases Controlled?

As with most software, OSS has multiple versions, releases, and security updates of which one's program is not in control. The need for life cycle configuration management is vital in ensuring system compatibility. A strategy needs to be developed to determine how one's program will handle periodic releases of the OSS software. Depending on the software, each release may require configuration, interface and installation, or system changes to remain compatible with the rest of the system.

What is the Maturity of the Open Source Community?

Similar to a standard commercial company, the maturity and size of the open source community can vary greatly. Open source projects can be started by a single developer who has made its source code available and gained additional support as the project grew, or by corporations who fund and assist in the development of the project. Open Office, an open source office suite, is sponsored by Sun® Microsystems and has other corporate contributors such as Google® and IBM®. The Open Office project contains 30,000 source files and 9 million lines of primarily C++ code, according to the Open Office website, and it contains many of the features included in Microsoft Office.

Many factors affect the maturity of the open source community supporting the project. Navica® has developed an Open Source Maturity Model®, which is freely available and will assist in the assessment of the open source project. The Open Source Maturity Model provides a variety of templates to assess different areas of the open source project such as documentation, integration, product software, professional

services, technical support, and training. Those items are then further decomposed to help assess each area of the open source project.

Do You Need to Modify the Source Code?

The major difference between proprietary software and OSS is the ability to view, modify, and distribute the application source code. Code modification may lead to some undesired effects on the life cycle of the system. Modifying the source code would force the program to keep a private copy that is different from the open source project's repository. That may work without issue for the initial release, but remember, just like proprietary software, OSS periodically releases new versions, patches, and upgrades. Once one breaks off from the primary project, he or she is now responsible for any upgrades and associated testing as the releases may not be compatible with the modified version.

Code modification may not be as easy as one might think. Take the Open Office project mentioned previously. If someone required a code modification and provided the development team with 9 million lines of code, a seemingly trivial modification may turn out to be a daunting task. Unfamiliarity with the application or programming language may cause additional complications. Most OSS uses a modular design so it can be easier to locate the code segment for which the modification is needed; however, the effects on the application may still be unknown.

One possibility is to make the modifications to the source code and submit the update to the OSS project's committee for review and possible incorporation within the next software release. If accepted, the update would go through the project's revision, testing, and review process during subsequent releases, and one would no longer need the old version of the software. Similar to most commercial software, the open source community does what is best for the community and not one's specific program. Therefore, there is no guarantee one's changes will be included in the next software baseline. As with any software application, when new functionality is added, the project is now responsible for maintenance, testing, and bug fixes for the added piece of functionality.

While modifications provide an added level of complexity, OSS does provide several alternatives over commercial software. One alternative may be deciding there is only a need to use a portion of the source code within the project. If the OSS is modular in design, it may be easy to extract only the functionality needed to incorporate into the application. That may be the best option if only a small piece of the OSS functionality is required. As with proprietary software, there is a point where "too much of a good thing" can turn bad. If one takes several pieces of different systems and includes them in his system, the system may become difficult to maintain, especially when each addition is in a different programming language, contains different interfaces, and may require additional dependencies. This can be exemplified by using a car analogy. Consider buying a Chevy Camaro but realizing that it will require the engine in the Ford Mustang and the electronics of the Audi A4. After integrating the required functionality of the other automobiles, the owner would have a system that met all of his requirements. However, if the vehicle needed maintenance, the owner would no longer be able to take it back to the Chevy dealership because a modification to the electronics system may adversely affect the engine because the components were not initially design to work together. In addition, if Audi releases an electronics upgrade, the owner may be unable to use the new software due to compatibility issues with the nonstandard engine.

Is OSS the Full Solution?

As with most proprietary products, OSS may not provide a solution that will satisfy everyone's requirements. Users may have to sacrifice functionality for a faster time to field. Gen. David Petraeus, commander of U.S. Central Command, recently said in an interview, "Never underestimate how important speed is." Additionally, he pointed out that in most cases, the soldiers are willing to accept an 80 percent solution. This is where constant user involvement is imperative in order to help make an informed decision. The user decides if less functionality provided sooner outweighs the time needed to develop the functionality from the ground up.

Conversely, OSS comes with a variety of features and could include many more features than are required by one's program. This inundation of extra features may require additional training, testing, and/or information assurance assessments to use the software in an operational environment. Removal of those features is also an option, but one must remember the risks mentioned in the modification section.

Does OSS Offer Maintenance and Support?

OSS may also contain a maintenance and support element that is available for a cost. MySQL offers an enterprise package that includes the software, support, and additional monitoring tools. Depending upon the needs of the program, one may consider a support package in which the cost would need to be added into the life cycle cost of the system.

Overall Evaluation of OSS

If one chooses to modify the source code and keep his own version, OSS can easily morph into government off-the shelf

software, losing most of the value of leveraging from the OSS community. At that point, the program becomes responsible for having developers available for maintenance and support. One may also find himself maintaining a great deal more features than what is required for the program. Most OSS projects make the executable (installer) available for download. If one were to only download the executable, he will be left with what is essentially a proprietary product but with the added benefit of having access to the source code. Modifying the source code may be a researcher's best option as long as he is prepared for the possible future consequences.

The items identified in this article are only a few of the considerations for evaluating OSS for use within a program. Other factors that may need consideration are security, prerequisites, reliability, and performance. The Defense Acquisition University Best Practices Clearinghouse (<https://bpch.dau.mil>) contains a forum to enable the sharing of best practices when evaluating OSS throughout DoD.

Remember, the open source community is available because projects make their source code available. Making someone's code available may allow for external reviews and could improve code quality. The Defense Information Systems Agency has developed an online open source repository at <www.forge.mil> called SoftwareForge. SoftwareForge hosts open source and community software projects within the DoD. If public availability it not an option, SoftwareForge may be a more secure alternative.

About the Author

Matthew Kennedy is a professor of software engineering at the Defense Acquisition University. He served in the U.S. Air Force as a network intelligence analyst and he has more than 10 years of experience in information technology. He has a bachelor's and master's degree in computer science.

This article first appeared in Defense AT&L, July-August 2010.

Open Source Software Is Commercial

By Dr. David A. Wheeler

Nearly all publicly-available open source software (OSS) is commercial software. Unfortunately, many government officials and contractors fail to understand this. This misunderstanding can result in higher costs, longer delivery times, and reduced quality for government systems. There are also legal risks: government officials and contractors who do not understand this, yet influence the selection or use of software, will probably fail to comply with U.S. law and regulations on commercial software. Finally, such government officials and contractors do not understand the modern commercial software marketplace, and thus are destined to make poor decisions about it.

This article explains ***why*** it is important to understand that OSS is commercial; explains why nearly all OSS is commercial software per U.S. law, regulation, and DoD policy; and shows why open source software is commercial even beyond the "letter of the law" because it has all the usual earmarks of commercial practice. But first, we must define the term OSS.

Defining Open Source Software

As the official DoD policy on OSS states, "*Open Source Software is software for which the human-readable source code is available for use, study, reuse, modification, enhancement, and redistribution by the users of that software*" [DoD2009]. Other definitions for OSS (also called Free Software, Free/Libre/OSS, and FLOSS) include the Free Software Foundations' "Free Software Definition" and the Open Source Institute's "Open Source Definition." Successful OSS is typically co-developed and maintained by people from multiple organizations working together. For general OSS information, see "Open Source Software (OSS) in U.S. Government Acquisitions" (Software Tech News, Vol. 10, No. 2) and [Wheeler2007].

Why is this important?

Acquirers (both government and contractors) risk much by failing to understand that OSS is commercial.

First, they risk ignoring the best possible alternatives that they are required by law to consider. As [DoD2009] attachment 2 part 2 notes, "Executive agencies, including the Department of Defense, are required to conduct market research when preparing for the procurement of property or services by 41 USC Sec. 253a … (see also FAR 10.001…). Market research for software should include OSS when it may meet mission needs." The Federal Acquisition Regulations (FAR) part 12 requires agencies to "Conduct market research to determine whether commercial items or nondevelopmental items are available that could meet the agency's requirements."

Second, they risk failing to comply with U.S. law and regulations that require preference for commercial software (see 10 USC 2377) and a maximal use of commercial software (where practicable). FAR part 12 states that agencies must "(b) Acquire commercial items or nondevelopmental items when they are available to meet the needs of the agency; and (c) Require prime contractors and subcontractors at all tiers to incorporate, to the maximum extent practicable, commercial items or nondevelopmental items as components of items supplied to the agency."

Finally, there is the risk of paralysis. There are many regulations and local rules about commercial items. Someone who doesn't realize that nearly all OSS is commercial won't know what rules to follow, and can become effectively paralyzed. Once they realize that nearly all OSS is commercial, they can usually follow the well-understood rules for commercial software.

OSS is commercial by law, regulation, and policy

The DoD policy on OSS [DoD2009] attachment 2 part 2 says, "In almost all cases, OSS meets the definition of 'commercial computer software' and shall be given appropriate statutory preference in accordance with 10 USC 2377 (reference (b)) (see also FAR 2.101(b), 12.000, 12.101 (reference (c)); and DFARS 212.212, and 252.227-7014(a)(1) (reference (d)))." We can confirm this by examining U.S. law and regulation.

U.S. law governing federal procurement (specifically 41 USC 403) formally defines the term "commercial item" (underlining added) as:

"*(A) Any item, other than real property, that is of a type customarily used by the general public or by nongovernmental entities for purposes other than governmental purposes, and that—*

(i) has been sold, leased, or licensed to the general public; or

(ii) has been offered for sale, lease, or license to the general public.

(B) Any item that evolved from an item described in subparagraph (A) through advances in technology or performance and that is not yet available in the commercial marketplace, but will be available in the commercial marketplace in time to satisfy the delivery requirements under a Federal Government solicitation.

(C) Any item that, but for—

(i) modifications of a type customarily available in the commercial marketplace, or

(ii) minor modifications made to meet Federal Government requirements, would satisfy the criteria in subparagraph (A) or (B).

(D) Any combination of items meeting the requirements of subparagraph (A), (B), (C), or (E) that are of a type customarily combined and sold in combination to the general public.

(E) Installation services, maintenance services, repair services, training services, and other services if—

(i) the services are procured for support of an item referred to in subparagraph (A), (B), (C), or (D), regardless of whether such services are provided by the same source or at the same time as the item; and

(ii) the source of the services provides similar services contemporaneously to the general public under terms and conditions similar to those offered to the Federal Government.

(F) Services offered and sold competitively, in substantial quantities, in the commercial marketplace based on established catalog or market prices for specific tasks performed or specific outcomes to be achieved and under standard commercial terms and conditions.

(G) Any item, combination of items, or service referred to in subparagraphs (A) through (F) notwithstanding the fact that the item, combination of items, or service is transferred between or among separate divisions, subsidiaries, or affiliates of a contractor.

(H) A nondevelopmental item, if the procuring agency determines, in accordance with conditions set forth in the Federal Acquisition Regulation, that the item was developed exclusively at private expense and has been sold in substantial quantities, on a competitive basis, to multiple State and local governments."

This definition in U.S. law is reflected in the Federal Acquisition Regulation (FAR) FAR 2.101, as well as the DoD FAR Supplement (DFARS) 212.212 and 252.227-7014(a)(1). The DFARS definition is shorter, but for our purposes has the same basic thrust.

Thus, OSS that has been released and licensed to the general public, and has at least one non-government use, is by definition commercial. Note that OSS that implements government functions, or was originally developed by the government, is still commercial as long as it meets this definition (e.g., it is licensed to the public and used for at least one non-government purpose). If the OSS isn't released yet, but will be in time, it is still commercial (this enables OSS "bounty systems"). The government can often pay for modifications to OSS (e.g., to address government-specific needs) and still consider the result commercial. Related services

(e.g., installation, repair, and training), even if they're from a different source than the original author, are also typically commercial per this definition.

Note that software often ends up being used for non-government purposes, even if it was originally developed for a government purpose. Software developers often work to make their software more general-purpose, so that they have more potential users. In addition, many organizations perform functions that are similar to functions performed by the government. For example, many governments need integrated library systems, but many other non-government organizations (such as large universities and companies) need them also.

> *"The lack of money changing hands in open source licensing should not be presumed to mean that there is no economic consideration"*

DoD's "Commercial Item Handbook" (November 2001) explains that the broadness of this government definition of "commercial item" is intentional, because it "enables the Government to take greater advantage of the commercial marketplace." The DoD policy memo "Commercial Acquisitions" (Jan. 5, 2001), Appendix A in the handbook, explains that the benefits of commercial item acquisition include "increased competition; use of market and catalog prices; and access to leading edge technology and 'non-traditional' business segments." Note that those who created these definitions and policies anticipated that there will be changes in the commercial market, including "non-traditional business segments."

This interpretation is supported by documents other than [DoD2009]. Department of the Navy memorandum "Department of the Navy Open Source Software Guidance" (signed June 5, 2007) was released specifically to make it clear that OSS is commercial. It says that the Navy will "treat OSS as [Commercial Off-the-Shelf (COTS)] when it meets the definition of a commercial item." OMB Memo M-03-14 "Reducing Cost and Improving Quality in Federal Purchases of Commercial Software" is about commercial software, and it specifically says that its SmartBUY initiative will include open source software support.

OSS is commercially developed and supported

OSS is commercial, even if we ignore US law and regulation. The *New York Times Everyday Dictionary* (1982) says that "commercial" means either (a) "oriented to profit-making," or (b) "of, pertaining to, or suitable for … [dealings, the buying and selling of commodities, or trade]." Let's start with the first definition.

Many for-profit companies make some or all of their money developing and/or supporting OSS, including Red Hat, IBM, Oracle, and others. *InformationWeek*'s David DeJean, in his article, "Is Open-Source A Business Model? $500 Million Says It Is," notes that Citrix paid $500 million for XenSource (maker of the OSS Xen hypervisor). IBM says that in 2001 it invested $1 billion in Linux, and that by 2002 it had already almost completely recouped that investment, suggesting some astounding returns on investment. *InfoWorld*'s Savio Rodrigues reported on July 10, 2007, that venture capitalists invested $1.44 billion in OSS from 2001 through 2006. Someone who uses "commercial" as the opposite of OSS will have trouble explaining why Red Hat is listed in the New York Stock Exchange (for example), since they focus on developing and releasing OSS.

For-profit organizations use or support OSS for many different reasons. Some give away the OSS and sell the support (such as training, customization, and support/ maintenance). Many use and support OSS as a support infrastructure for the product or service they actually sell, i.e., for cost avoidance by cost sharing. Many for-profit organizations have realized the value of "commoditizing your complements," that is, you'll sell more of your product if things related to it (that you don't sell) are cheaper.

Once you use the second broader definition of "commercial," it is even clearer that OSS is commercial. Economists often emphasize the difference between wealth and money. Some OSS projects attempt to earn money (directly or indirectly), but nearly all OSS projects attempt to create wealth in the form of improved software. They attempt to create wealth via trade and dealings ... a fundamentally commercial notion.

OSS developers give their users many more rights than proprietary products do, typically with the expectation that others are thus likely to contribute back to the project. Thus, most non-profit OSS projects are actually trying to achieve financial gain – it just happens that they are trying to receive gains of additional or improved software instead of money. As Linux kernel creator Linus Torvalds noted in a 2003 letter

to SCO, the U.S. Code Title 17, Section 101 (the law that creates and defines copyrights in the U.S.) explicitly defines the term "financial gain" as including "receipt, or expectation of receipt, of anything of value, including the receipt of other copyrighted works." Thus, while OSS projects may not receive money directly, they typically do receive something of value in return. Ganesh Prasad's "How Does the Capitalist View Open Source?" captured this concept nicely in May 2001.

The U.S. Court of Appeals for the Federal Circuit formally stated that there are economic considerations with OSS. In their ruling on Jacobsen v. Katzer (August 13, 2008), they said that "*Open Source software projects invite computer programmers from around the world to view software code and make changes and improvements to it. Through such collaboration, software programs can often be written and debugged faster and at lower cost than if the copyright holder were required to do all of the work independently. In exchange and in consideration for this collaborative work, the copyright holder permits users to copy, modify and distribute the software code subject to conditions that serve to protect downstream users and to keep the code accessible... Traditionally, copyright owners sold their copyrighted material in exchange for money. The lack of money changing hands in open source licensing should not be presumed to mean that there is no economic consideration, however. There are substantial benefits, including economic benefits, to the creation and distribution of copyrighted works under public licenses that range far beyond traditional license royalties. For example, program creators may generate market share for their programs by providing certain components free of charge. Similarly, a programmer or company may increase its national or international reputation by incubating open source projects. Improvement to a product can come rapidly and free of charge from an expert not even known to the copyright holder. The Eleventh Circuit has recognized the economic motives inherent in public licenses, even where profit is not immediate....*"

Also, note that many OSS developers are now well-paid for their work. Consulting company Bluewolf found that "the advancement of open source software is triggering an increasing need for specialized application developers ... higher-end, more complex application development proves difficult to complete overseas ... The rise of open source software in application development puts developers with a specialization in those technologies in a position to ask for a 30 or 40 percent pay increase..." [Eddy2008]. Provably 70% of all Linux kernel development is by developers who are being paid to do this work [Corbet2010], and the actual figure is probably much higher.

Alternatives

The most common antonym for OSS is "proprietary software;" other terms include "closed source," "non-Free," "non-OSS," and "non-FLOSS." I tend to use "proprietary software" as the antonym, simply because it seems to be the most widely used and thus better understood. Do not call OSS *non-commercial*, because nearly all OSS *is* commercial.

Conclusions

It's time to end the nonsense. OSS is practically always commercial, which means that there are two major types of commercial software: proprietary software and OSS. Terms like "proprietary software" or "closed source" are plausible antonyms of OSS, but "commercial" is absurd as an antonym, and phrases like "commercial or OSS" make no sense.

This has real-world implications. In particular, government acquisitions (including work performed by contractors) must include OSS in their market research and must carefully consider OSS candidates.

Bibliography

[Corbet2010] Corbet, et al. December 2010. "Linux Kernel Development" http://www.linuxfoundation.org/docs/lf_linux_kernel_development_2010.pdf.

[DoD2009] DoD CIO. 2009-10-19. "Clarifying Guidance Regarding Open Source Software (OSS)" http://cio-nii.defense.gov/sites/oss/2009OSS.pdf.

[Eddy2008] Eddy, Nathan. 2008-02-26 "Report: Open Source Adoption Increases App Dev Pay," *ChannelWeb*.

[Wheeler2007] Wheeler, David A. 2007-04-16, "Why OSS/FS? Look at the Numbers!" http://www.dwheeler.com/oss_fs_why.html.

[Wheeler2009] Wheeler, David A. "Free-Libre / Open Source Software (FLOSS) is Commercial Software," revised 2009-02-03. http://www.dwheeler.com/ essays/commercial-floss.html. Summary published as "F/LOSS is Commercial Software," *Open Source Business Resource*, Feb. 2009, pp. 25-33.

This article is based on [Wheeler2009].

The publication of this paper does not indicate endorsement by the Department of Defense or IDA, nor should the contents be construed as reflecting the official positions of those organizations.

Implementing Open Standards in Open Source

By Lawrence Rosen

Industry standards morph into functional computer software. I use the word "morph" on purpose to avoid any term that can be found in US copyright or patent law. Morphing is a special effect in motion pictures and animation to turn one image into another through a seamless transition. Wikipedia shows an image of George W. Bush morphing into Arnold Schwarzenegger, and so too the morphing of an industry standard into software can result in something that looks entirely different at an expressive level and that potentially does useful things.

In the case of software industry standards, morphing transforms a written specification into working code through a mental process conducted internally by programmers and engineers. The end result – functional software – is a created outcome of human intellect that starts with a written specification and ends with a working implementation.

For attorneys, software specifications are unusual beasts. A specification may be the description of something patentable, but it is not itself patentable. Only an implementation of a specification, something that can be made, used, or sold, may be subject to patent infringement lawsuits (35 USC 271). Likewise, a specification itself can also be copyrighted, although the copyright does not extend to any idea, procedure, process, system, method of operation, concept, principle, or discovery, regardless of the form in which it is described, explained, illustrated, or embodied in the specification (17 USC 102(b)). The rights to intellectual property in an industry specification (and thereby perhaps control over its intellectual content) are thus subject to some difficult legal questions with uncertain answers.

This topic gained added relevancy recently because of the patent and copyright infringement lawsuit by Oracle against Google relating to Java. We have been led to believe that Java is an industry standard for a programming language. In compliance with the rules of the Java Community Process ("JCP"), the Java community develops final specifications for technology to be included in the Java platform and publishes free implementations of those specifications under open source licenses. There are currently more than 300 published specifications for the Java language. There is much free and open source software implemented in compliance with these Java specifications. Many companies, individuals, and non-profit foundations (including the Apache Software Foundation of which I am a member) participate in the JCP with the goal that those specifications be available to all free of charge. The *Oracle v. Google* lawsuit has placed a patent and copyright cloud on Java specifications and software.

Specifications are different from software, but they are weapons in the competitive software wars and they are subject to legal control by contract and by law. Companies try to control specifications because they want to control software that implements those specifications. This is often incompatible with the freedom promised by open source principles that allow anyone to create and distribute copies and derivative works without restriction.

This article explores ways that are available to compromise that incompatibility and to make open standards work for open source.

Copyright on Industry Standards: Does Implementation Create a Derivative Work?

The standard of proof for copyright infringement includes the element of "substantial similarity." Because copyright protects expression, an infringing work must resemble the expression of the original, not merely its underlying ideas and concepts. If they do not look alike, it will be difficult to prove that an implementation is a derivative work of a specification. In the *SCO v. IBM* litigation[1], for example, the plaintiff was ultimately unable to identify any specific code in Linux that was derivative of its own UNIX software, even though they were functionally similar.

Often a specification and its implementation are very dissimilar. Specifications that contain English words describing functionality are intended to be implemented in C++ or Java or some other programming language, and the natural language description – certainly to the untrained eye – doesn't resemble

1 Editor's Note: On March 6, 2003, the SCO Group (formerly known as Caldera Systems) filed a $1 billion lawsuit in the US against IBM for allegedly "devaluing" its version of the UNIX operating system. The amount of alleged damages was later increased to $3 billion, and then $5 billion. SCO claimed that IBM had, without authorization, contributed SCO's intellectual property to the codebase of the open source, Unix-like Linux operating system. Source: Wikipedia

"Innovation without protection is philanthropy"

- Mark Blaxill, The Invisible Edge, 2009

the resulting code in the slightest. The judges and juries that will determine copyright infringement won't be able without expert advice to determine the substantial similarity of highly technical expressive works of software.

Some software experts view this dissimilarity of expression between specification and software code as a technical disadvantage, and so they are trying to create specifications that are their own software implementations. In the HTML5 project, for example, which is creating standards for structuring and presenting content in browsers on the World Wide Web, the drafters are experimenting with a specification technique that replaces specification text with public domain reference implementations in terms of an abstract state machine, in an attempt to improve compatibility by avoiding the imperfect conversion of English to source code. Implementers are encouraged to copy specification text as software. In these cases, at least, a software implementation is obviously a derivative work of the specification, and thus an infringement unless licensed.

Regardless of the style of specification writing and the programming language, it is to nobody's advantage to argue in court whether an implementation is a derivative work of the specification. A legal argument about whether an implementation is a derivative work, no matter how convincing, is not as convincing as a written license that expressly authorizes those derivative works. And so implementers seek licenses and standards organizations offer them. Standards bodies have devised their copyright licensing policies with the expectation that software implementations will be derivative works of their specifications.

Because software implementations will often be distributed under open source licenses, the compatibility of specification copyright licenses with open source rules becomes critical. The Open Web Foundation (OWF), in its specification license, addresses the copyright issue for software specifications succinctly and directly:

I grant to you a perpetual (for the duration of the applicable copyright), worldwide, non-exclusive, no-charge, royalty-free, copyright license, without any obligation for accounting to me, to reproduce, prepare derivative works of, publicly display, publicly perform, sublicense, distribute, and implement the Specification to the full extent of my copyright interest in the Specification.

Note that the copyright license here is "to the full extent of my copyright interest," thus effectively ignoring the question posed in the title of this section. Note also that the broad copyright grant to implement the Specification is fully compatible with all open source (and proprietary) licenses.

That copyright grant, however, is not satisfactory to some standards organizations. It goes too far with respect to allowing derivative works of a specification *as a specification*. These standards organizations seek to protect the purity of their specifications by forbidding other standards

organizations to take over those specifications and modify them. In colloquial terms, and as they frequently justify this restriction, these organizations wish to prevent the "forking" of their specifications that might result in incompatible implementations.

This is a reasonable business and technical concern for a standards organization. Anyone who witnessed the early years of the browser software wars will remember how functionally incompatible browsers inhibited the development of advanced websites, and allowed commercial competition rather than technical benefits to dictate browser functionality. No software implementer wants to repeat that experience.

Compatibility requirements, however, are anathema to open source implementers. A requirement that all implementations function in a particular way is contrary to every open source license that guarantees complete freedom to create derivative works. The desire of standards organizations to prevent forking of open standards contradicts the requirement of open source licenses that permit any derivative works.

This problem has recently been the subject of heated discussion in W3C relating to the new HTML5 specification. Nearly 80% of the W3C members responding to a survey said they do not want W3C to permit forking of W3C specifications, but they also overwhelmingly say that they want to encourage implementation of any open source software. By way of compromise, one proposal was for a new HTML5 license to allow software derivative works but to forbid specification derivative works. The Free Software Foundation, however, has argued that this restriction means that the proposed license is incompatible with the GPL. As this paper is being written, no final decision has been made about this license.

This problem was addressed by IETF in yet another way. They require specification writers to distinguish between "text" and "code." The IETF copyright license allows derivative works of the code but not the text portions of its specifications. Thus implementers may use the code portions, but if they seek to document those code portions they are not allowed to create derivative works of the IETF specification text. This rather arbitrary way of addressing copyright law issues of derivative works of specifications is based on distinctions between text and code that are not found in copyright law or in computer science. I do not believe this distinction is enforceable practically.

The *Oracle v. Google* lawsuit presents another aspect of this derivative work copyright problem. Oracle asserts in its complaint that Google has infringed Oracle's Java copyrights (presumably relating to the Java specifications, although the complaint in that case is not clear). Java specifications are published under a license that requires implementers to validate compliance with Java specifications using test compatibility kit (TCK) software licensed by commercial companies with contractual restrictions on the types of software derivative works that can be implemented. As such, it is incompatible with the requirements of the Apache License under which the Apache Software Foundation, and Google, publish their software. The Apache Software Foundation objected publicly to Sun, the Java specification steward, when it first imposed these contractual restrictions on the kinds of software derivative works that can be created if open source implementers license the Java TCK; now Oracle is the Java steward, and the concern has been reanimated by this recent litigation.

This question of whether software is a derivative work of a specification has thus become more important recently. Indeed, there may be no single answer that would satisfy those who create and seek to protect standards and those who implement those standards under open source licenses. Agreement on specification licenses will require a degree of compromise over copyrights that neither commercial companies nor the open source community have yet achieved.

we like your feedback

At the DACS we are always pleased to hear from our Software Tech News magazine readers. We are very interested in your suggestions, compliments, complaints, or questions. Please visit our website ***softwaretechnews.com****, and fill out the survey form. If you provide us with your contact information, we will be able to reach you to answer any questions.*

Patents on Industry Standards: Can a Specification Infringe a Patent?

Copyrights are not the only intellectual property problem besetting software standards. Far more difficult are the effects of patents on the software ecosystem, because copyrights encumber only derivative works but patents can encumber *any* implementation. A clean room implementation of a specification will avoid copyright infringement -- but it is not so easy to avoid patent infringement.

Patents are not a problem for specification writers, but rather for implementers. Anyone is free to write a description of how to perform a process or method – indeed the patent system requires the open publication of just such a specification when a patent is granted. But when that specification is transformed into functioning software and distributed for actual use by actual people and companies, those users may be patent infringers. In the proprietary software world such risks are commonly borne by the vendors of software products with offers of limited warranties and indemnity, usually capped at reasonable dollar limits. But with free and open source software that is distributed without warranties or indemnities, the risk of patent infringement when using implementations of standard specifications is borne by the user.

Note that this patent risk is in practice quite low. While hypothetical situations can be litigated easily in the mind, there have been no notable patent infringement lawsuits in U.S. federal court over software standards implemented in open source software. When major software companies cooperate openly to develop industry standards in organizations like W3C and IETF, few of them have much incentive to demand royalties or impose burdens on competing implementations. Some standards, of course, such as those for music and movie distributions on the web, are proprietary and licensed for a fee. Open source projects have typically refused to implement those encumbered standards, which may partly explain the dearth of infringement lawsuits. But when the stated goal of the standards setting organization is a royalty-free patent license for open source implementations, litigation over such patents is unnecessary.

The *Oracle v. Google* case is a recent exception. As I mentioned earlier in the copyright context, this lawsuit relates to Java software. Java is an industry standard for a popular programming language, and there are several proprietary and open source implementations of that language and its related programming libraries. Given the widespread reliance of the software industry on Java, it was a surprise when Oracle asserted a number of its patents (acquired in its acquisition of Sun) against Google, purportedly for Google's implementation of the Android open source operating system.

It is far too early in this litigation to comment on the merits of the case, but the effect of the *Oracle v. Google* lawsuit has already been to put a cloud on the Java standard. This litigation is likely to discover some hitherto unexplored areas of software competition practices that are problematic given the explicit promises and community expectations of the parties to the Java Community Process under which Java was developed. In particular, the Apache Software Foundation has already complained about the JCP requirement that implementers acquire and pass a Test Compatibility Kit (TCK) before receiving Java patent licenses from Oracle (Sun), when those TCK licenses expressly prohibit certain kinds of implementations and derivative works. That TCK licensing practice for industry standards is not compatible with open source, and those contractual restrictions on the use of specifications have not be accepted by Apache.

Traditional standards organizations have finessed the patent problem by requiring authors of specifications to offer Reasonable and Non-Discriminatory (RAND) licenses. RAND doesn't mean "free", however, and it doesn't mean "without encumbering conditions," and so RAND promises are of little use to open source implementers. Fortunately, because open source implementations have become important as validations of open standards, most software standards are now actually published under RAND-Z, or zero-priced, RAND licensing terms. This promise of reasonable and non-discriminatory licensing terms is often made but not usually put into explicit words. For example, most contributions to IETF are accompanied by RAND promises, but actual licenses are not published anywhere on the IETF website. Nor does the W3C royalty-free patent policy require the publication of an actual patent license, although members of W3C are committed by their membership agreement to grant such a license if asked. Nobody asks.

Open source licensing practices no longer tolerate such ambiguities. Contributors to mature open source projects sign Contributor License Agreements to provide written confirmation of their copyright and patent promises. The Open Web Foundation (OWF) is drafting such explicit agreements for industry standards to mitigate the risks of patent litigation.

One important difference between the OWF proposal and traditional industry standard practices is in the identification

of the patent claims being licensed for free. In most standards organizations the patent grant is to "Necessary Claims" – meaning those patent claims that are necessary to implement the specification without infringing. Actual terms differ; sometimes the rule is "actual technical impossibility" of implementation without a patent license and sometimes merely "financial or technical impracticality" of implementation unless a patent license is available. Either way, this Necessary Claims language means that if there are multiple ways of implementing a specification, then no patent claim is a Necessary Claim.

OWF takes a more generous view of patent licensing for industry standards. Its patent grant says simply:

The Promise. I, on behalf of myself and my successors in interest and assigns, irrevocably promise not to assert my Granted Claims against you for your Permitted Uses, subject to the following.

The OWF agreements then define "Granted Claims" simply as those claims that are infringed by "Permitted Uses":

"Permitted Uses" means making, using, selling, offering for sale, importing or distributing any implementation of the Specification 1) only to the extent it implements the Specification and 2) so long as all required portions of the Specification are implemented. Permitted Uses do not extend to any portion of an implementation that is not included in the Specification.

This means that, so long as he is implementing the required portions of a specification, and even if there are multiple ways of implementing that specification, an implementer is free to choose a patented method if it is better or more efficient. The license extends only to the desirable goal of implementing the specification, however, and it is not a license for all uses of the Granted Claims.

To protect patent owners from unanticipated grants of their patents, as well as to protect implementers from game-playing by patent owners, the OWF agreements define Granted Claims as follows:

"Granted Claims" are those patent claims that I own or control, including those patent claims I acquire or control after the Date below, that are infringed by Permitted Uses. Granted Claims include only those claims that are infringed by the implementation of any portions of the Specification where the Specification describes the functionality causing the infringement in detail and does not merely reference the functionality causing the infringement.

Under the OWF, patent owners are thus able to read a Specification to determine which of their patents will be licensed to implementers, without needing to determine whether their patent claims are in some vague sense Necessary Claims because there are no alternatives. And implementers are able to read a Specification and, as long as they are implementing all the required portions of the Specification, they needn't worry about patent litigation for that implementation.

Conclusions and Predictions

It is not yet known whether the OWF provisions for patent licensing will overcome the resistance to change for IP policies in standards organizations in order to make both copyrights and patents freely available to open source implementers of open standards. Nor is it known what effect the *Oracle v. Google* lawsuit will have on the Java standards and the future expectations of implementers to be free to create software based on open standards. Intellectual property attorneys live in interesting times.

In a way, this is very much like the challenges facing Creative Commons when it found chaos and uncertainty in the licensing of music and art and film for free use by all. Open standards are equally fundamental to the ways we live. That is why implementers ought to be free of copyright and patent restrictions to create the open source software on which our world depends. And that is also why attorneys should understand carefully the intellectual property obligations of contributors to and users of open standards.

About the Author

Lawrence Rosen is partner of Rosenlaw & Einschlag (www.rosenlaw.com), a technology law firm that specializes in intellectual property protection, licensing and business transactions for technology companies. He currently advises many open source companies and non-profit open source projects, including as member of and counsel to Apache Software Foundation and as a member of the board of directors of Open Web Foundation. He is also serving at W3C on the Patents and Standards Interest Group and on the New Standards Task Force. Larry's book, *Open Source Licensing: Software Freedom and Intellectual Property Law*, was published by Prentice Hall in 2004. He is a Lecturer in Law at Stanford Law School.

Running Open Technology Development Projects

By John Scott, Dr. David A. Wheeler, Mark Lucas, and J.C. Herz

"How to get started" is a question continually asked. This article lays out the basic framework for running an open technology development (OTD) military focused project. The first section describes how to establish an OTD program once a project proposal has been accepted. The next sections discuss establishing a technical infrastructure for collaboration, communication issues, technical management/technical criteria, and continuous delivery. Much more information on how to do this from an open source software (OSS) project perspective can be found in chapter 2 of [Fogel2009].

Step 1: Determine reuse options

First, search for existing OSS projects that have relevant functionality. A simple web search of the string "open source software" plus a desired capability will often turn up something close to what you need. Also review OSS repositories sites such as http://www.sourceforge.net, http://www.freshmeat.net, http://www.github.com, http://directory.fsf.org and http://code.google.com. Even if there is nothing available to use directly, there might be piece-parts that can be integrated or useful ideas.

Opportunistic adoption of OSS is important because technological innovation is primarily occurring on the unclassified internet, not within the military sphere. Most of the piece-parts for any given project are already out there, and there is an expanding wave front of OSS software that can rapidly advance the needs of government projects. Careful evaluation, selection, and participation in these external projects is the most effective way to evolve capabilities over the life cycle of a government program. Existing Government Off The Shelf (GOTS) software may quickly become obsolete once there is a public Commercial Off The Shelf (COTS) project (including an OSS project) with the same goal.

If you have software that was previously developed as part of a government contract, determine if you have sufficient intellectual rights to release or transition the software as an OTD project. Many government programs have existing technology that was originally funded by the government. If the intellectual rights over those technologies is inadequate or cannot be determined, the government should consider negotiating with the appropriate integrators/vendors to release the source code under less restrictive data rights sufficient for an Open GOTS (OGOTS) or OSS project. An easy way to do this is to simply fund the conversion process for the contractor(s).

Step 2: Identify the Projects to be Established

Given the reuse options, identify what new projects are necessary and which existing projects need to be transitioned to OTD. In some cases, the "new project" may be a project to extend some existing OTD project and get that extension integrated into the original project. Where possible, split up the project into several smaller projects with clear interfaces. These smaller projects may be divided according to various criteria, including the likelihood of reuse (to maximize the number of participants in at least some of the projects) and the need to limit access (classified or export-controlled modules may need to be separated from other components, e.g., by creating an unclassified "framework" into which controlled "plug-ins" can be placed).

Name each project so that is not easily confused with other projects. It should be pronounceable and easy to find on a web search (ideally, it would be the only result from a search; certainly avoid unsearchable names like "the" or "why").

Each new project (including any existing project transitioning to OTD) needs a statement of intent that references the OTD software maintenance philosophy. As recommended in [Fogel2009] "the mission statement should be concrete, limiting, and above all, short." The mission statement should make it clear that the goal is to use open development principles (e.g. avoiding lock-in to a single supplier) and what the resulting products should do. Here's an example of a good one, from http://www.openoffice.org:

> *To create, as a community, the leading international office suite that will run on all major platforms and provide access to all functionality and data through open-component based APIs and an XML-based file format.*

In a DoD project, the software maintenance philosophy

statement might reference DFARS 227.7203-2 ("Acquisition of noncommercial computer software and computer software documentation"), and in particular the text at DFARS 227.7203-2(b)(1) (bold and underlining added):

> *Data managers or other requirements personnel are responsible for identifying the Government's minimum needs. In addition to desired software performance, compatibility, or other technical considerations,* ***needs determinations should consider*** *such factors as multiple site or shared use requirements, whether the Government's* ***software maintenance philosophy will require the right to modify or have third parties modify the software****, and any special computer software documentation requirements.*

Determine, for each project, whether it must be limited to only DoD or general government access as an OGOTS project. By default, projects should become COTS OSS instead of OGOTS. In some cases (e.g., due to classification or export control) a project must be limited to DoD or U.S. government access. GOTS projects present a higher risk than COTS projects, because by definition there are fewer potential contributors (decreasing competition and potentially increasing cost), and contractors (other than their copyright owners) are disincentivized from using GOTS projects because they cannot reuse those components or knowledge about them in other commercially viable ways. In many cases it is possible to split the project into two projects, one that is OSS (e.g., a "framework") and one that is OGOTS (e.g., a "plug-in" to the framework).

Step 3: Choose and Apply a Common License

Each project must have a clear and simple license that enables legal collaboration. A license lays out the rights and responsibilities of software developers and users. If the project is to be an OSS project, be sure to choose a well-known pre-existing OSS license, one that has already been widely certified as being OSS. It should be General Public License (GPL)-compatible, as the GPL is the most common OSS license. If the software pre-exists, it is usually wise to include its previous license as one of the options.

Step 4: Establish Governance

Projects that use OTD need to be governed. The governance process for each project needs to encourage collaborative development, but it must also allow the rejection of contributions where warranted. The OTD governance process must enable multiple organizations to work together to improve each component undergoing shared development (including its software, tests, and documentation), instead of re-developing separate independent components with similar functionality. Before discussing different governance models, it is important to note that *forkability* is necessary, as described next.

Forkability: A *fork* is a competing project established using a copy of an existing project's software.

It is critically necessary that an OTD project be *forkable.* That is, it must be possible to create a viable competing project using a copy of the existing project's software source code. Creating a fork is similar to a call for a "*vote of no confidence*" in a parliament. The fork creator is essentially asking developers and users to stop supporting the original project, and support the new forked project instead (supporting both projects is typically impractical over time).

Forks can also occur because the existing community doesn't plan to include a feature set part of the community deems important, reasons could include: support for a different operating system or middleware or inclusion of a new programming language. Whatever the reason, every effort should be made to keep forked projects somewhat as coordinated as possible.

Forkability is a necessary part of OTD governance. As long as a project is forkable, project leadership will strive to be responsive to users and developers. This is because if leadership decisions are particularly egregious, a forked project can be started under more responsive stewardship. Easy forkability actually reduces the risk of a fork, because leadership will be forced to listen to users and developers (because if they do not, a viable fork will emerge). In addition, easy forkability increases the likelihood of contributions; easy forkability provides significant protection to would-be contributors, because if they later disagree with project governance, they can create a fork.

Regardless of the governance model, the decision-maker(s) must avoid making a decision between alternatives too soon. If there is a disagreement, there may be a compromise or alternative approach that would be better than the immediately-obvious options. Therefore, decision-makers should try to get parties to find those compromises and alternatives. However, if a reasonable compromise cannot be found and a decision must be made, the decision-maker(s) should make that decision

after listening to all sides. That decision should be announced clearly, along with sound rationale. The decision-maker(s) must also be willing to change a decision given important new information, new options, or a change in circumstances.

A key to any governance approach is that the project must be forkable. Any governance model can eventually fail if the decision-makers have no need respond to others. If the project is forkable, then the leadership (regardless of the governance model) must in the end respect the needs of users and developers. More information can be found in [Fogel2009] chapter 4 and [Bacon2010] chapter 8.

Step 5: Establish Collaboration

Establishing collaboration isn't the same as creating a one-way communications strategy. Collaboration involves an easy interchange of ideas among many perspectives (including industry, academia and other government agencies offices and labs) to produce a better result than any one of them could have achieved separately.

When opening a formerly closed project, be sensitive to the magnitude of the change. Ensure that all its existing developers understand that a big change is coming. Explain it, tell them that the initial discomfort is perfectly normal, and reassure them that it's going to get better. Work to counter lapses into private discussions between long-time developers, and encourage their migration to community forums such as mailing lists. [Fogel2009]

Since some people will struggle with the openness of an OTD project, it is important to stress the need for openness. Point to guidance such as the current administration guidelines and mandates on transparency, and on the DoD 2009 memo on open source software which mandates that software be treated as data and shared appropriately. To quote the 2009 memo:

> "Software source code and associated design documents are "data" as defined by DoD Directive 8320.02 (reference (h)), and therefore shall be shared across the DoD as widely as possible to support mission needs."

There are of course discussions that must be kept closed to the public, such as company source selection and company proprietary data. But every attempt should be made to open up the software development process as much as possible. To simplify governance, the preferred method is to use an OSS license unless national interest dictates otherwise. The government should also require contractors and software integrators to organize their projects so that they are continuously transparent and open to the government for remote inspection.

Step 6: Create Project Technical Direction

For each project, determine key technical issues, such as which major components will be reused, what components the system must interact with, how it will be implemented (such as what implementation languages to use), what platforms it must work on, and basic developer guidelines.

Each project should stress *modularity*. A modular system is a system built from smaller interacting projects that can be developed in parallel and individually replaced without affecting other components. Modularity is key and simplifies technology and software IP reuse, eases and separates classification and export control issues, simplifies management, speeds deployment, reduces maintenance costs, and increases agility. A great military reference to modularity can be found at http://www.acq.osd.mil/osjtf/docsmemo.html. Well-known design patterns and architectural patterns can be used to divide problems into smaller components [Martin2000].

Step 7: Announcing

When a project is established and presentable (not perfect), or a significant event such as a major release occurs, tell others who would want to know. If you know of mailing lists where an announcement of your project would be on-topic and of interest, then post there, but be careful to make exactly *one* post per forum and to direct people to your project's own forums for follow-up discussion. If there are related projects (e.g., ones that might likely use it or be impacted by it), be sure to provide them the news, and invite them to post web links to your project website. Post an update on Intellipedia and the DoD Techipedia (this is especially important for OGOTS projects, since it can be difficult to find them if they are not publicly known). If it is a public OSS project, submit such announcements to freshmeat (http://freshmeat.net/).

Continuously Review Steps 1-7

Steps 1-6 should are the start of a continuous process where projects should constantly be cycling through the search for new components, growing the community, maturing the technologies and seeking to scale the size, heft and maturity

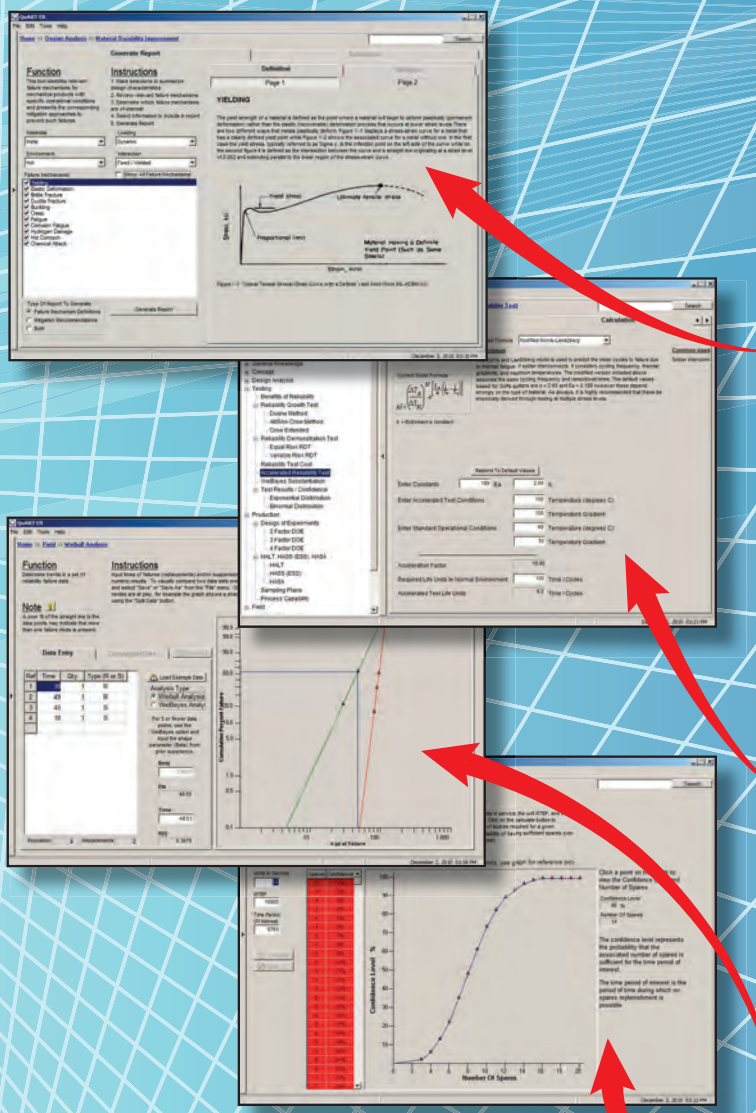
Continued on page 30

Introducing...

...the newest member of the Quanterion Automated Reliability Toolkit family...

QuARTER
Enhancing Reliability
Quanterion Automated Reliability Toolkit

››› With over two dozen new or improved tools!



		QuART	QuART PRO	QuART ER
		$99	$189	$399
General Knowledge	Reliability Advisor	No	Yes	Yes
	Reliability Program Cost	No	Yes	Yes +
	Reliability Imp rovement Cost	No	Yes	Yes
	Warranty Cost	No	Yes	Yes
	Statistical Distributions	No	Yes	Yes +
	Checklists	No	Yes	Yes
	Definitions	Yes	Yes	Yes
	Acronyms	Yes	Yes	Yes
Concept	Reliability Potential	Yes	Yes	Yes
	Reliability Allocation	No	No	Yes
	Reliability Tailoring	Yes	Yes	Yes
	Reliability Approach Assessment	No	No	Yes
Design Analysis	Derating	Yes	Yes	Yes
	Thermal Design	No	Yes	Yes
	Failure Modes	No	Yes	Yes +
	FMECA Worksheets	No	No	Yes
	Material Durability Improvement	No	No	Yes
	Parts Count Analysis	Yes	Yes	Yes
	Software Reliability Prediction	No	No	Yes
	Redundancy Modeling	Yes	Yes	Yes +
	Reliability Adjustment Factors	Yes	Yes	Yes
	Interference Stress Strength Analysis	No	No	Yes
	Availability Calculator	No	No	Yes
Testing	Benefit of Reliability	No	No	Yes
	Reliability Growth Testing - Duane Method	Yes	Yes	Yes
	Reliability Growth Testing - AMSAA-Crow Method	No	No	Yes
	Reliability Growth Testing - Crow Extended Method	No	No	Yes
	Reliability Demonstration Test Equal Risk RDT	Yes	Yes	Yes
	Reliability Demonstration Test Variable Risk RDT	No	No	Yes
	Reliability Test Cost	Yes	Yes	Yes
	Accelerated Reliability Test	Yes	Yes	Yes +
	WeiBayes Substantiation Testing	No	No	Yes
	Test Results / Confidence Exponential Distribution	Yes	Yes	Yes
	Test Results / Confidence Binomial Distribution	No	Yes	Yes
Production	Design of Experiments 2 Factor DOE	No	No	Yes
	Design of Experiments 3 Factor DOE	No	No	Yes
	Design of Experiments 4 Factor DOE	Yes	Yes	Yes
	HASS - ESS	No	Yes	Yes
	HASA	No	Yes	Yes
	Sampling Plans	No	No	Yes
	Process Capability	No	No	Yes
Field	Weibull Analysis	Yes	Yes	Yes +
	Optimal Replacement Interval	No	No	Yes
	Sparing Analysis (Graphical)	Yes	Yes	Yes
	Spares Analysis (Tabular)	Yes	Yes	Yes +

+ indicates improved features or functions

Download for only $399

QUANTERION
SOLUTIONS INCORPORATED

quanterion.com
(315)-732-0097/(877) 808-0097

Affordable tools for improving reliability…

Quanterion Solutions in a team member of the operation of the Reliability Information Analysis Center (RIAC)

This is a paid advertisement.

of the community. Over time the community should grow, thereby bringing in new people and ideas and leading to an increase in competence and competition for government contracts.

OTD Rules of the Road: Don't Fork OSS Solely for Government Use

A common mistake made by government projects that begin to adopt OTD approaches is to start with creating a fork by taking a snapshot of the source code and modifying it for their own needs, in isolation from the community surrounding that code.

This is a mistake because successful OTD projects are constantly evolving and improving. Creating a fork isolates all fork users from the main OTD project, including the improvements it makes. Refreshing OTD components is a very effective way of evolving the baseline for the project. It is important to remain synchronized with latest formal releases of the selected projects for system reliability, technological relevance, and obtaining the maximum benefit of an OTD approach.

In some cases, there is no need to modify the component itself. The component's application programmer interface (API) or plug-in system may provide the necessary flexibility without changing the component at all.

If a component must be changed, fixes and key enhancements to the baseline should be developed in consultation with original project and then submitted back to the original project. Unique government interfaces and functionality should be segregated through plug-in mechanisms or with APIs at a higher level. Taking this approach allows the government project to painlessly upgrade when new releases are made by the external project. Most useful components are continuously improved, so the ability to perform periodic upgrades must be built into the development and maintenance process.

In some cases, a project must make significant modifications to an OTD component it will depend on. First make sure that this is really the case; sometimes it is not. But if it is the case, discuss with that component's project the changes that need to be made, and look for ways to submit those changes incrementally to the upstream project. This will increase the likelihood that these changes will be accepted by that component's project. It is best if there is a contract incentive that changes to external projects be accepted back into those projects, to encourage the contractor to work with those external projects.

Open Standards

Use open standards. For purposes of this paper, an "open standard" is a specification that at *least* meets the European Union's definition as adopted in the European Interoperability Framework:

- The standard is adopted and will be maintained by a not-for-profit organization, and its ongoing development occurs on the basis of an open decision-making procedure available to all interested parties (consensus or majority decision etc.).
- The standard has been published and the standard specification document is available either freely or at a nominal charge. It must be permissible to all to copy, distribute and use it for no fee or at a nominal fee.
- The intellectual property - i.e. patents possibly present - of (parts of) the standard is made irrevocably available on a royalty-free basis.
- There are no constraints on the re-use of the standard.

Sometimes extensions are needed, but they should only be used with consideration as it can be easy to become accidentally locked into a proprietary extension. Being locked into a proprietary extension can be a problem, particularly if it is only implemented by a proprietary program (since this effectively eliminates competition, raising costs long-term). Consider requiring tests (as part of the contract) with an alternative implementation of a standard to increase the likelihood of staying within standard. Where appropriate, create or work to extend open standards.

Continuous Delivery

Development should be a continuous evolution through relatively small tracked changes. That way, others can effectively review these changes. These changes should not prevent a system from building or running. In some cases, a change will not have a user-visible effect, e.g., it may be an architectural change to prepare for future functionality. Daily builds followed by automated regression tests are highly recommended; these make problems immediately apparent.

Managing Intellectual Rights

Ensure that each contribution includes the necessary intellectual rights (including "data rights") that enable

the project developers and users to continue in their use, modification, and redistribution as appropriate. In particular, examine copyright markings on contributions, and look for the insertion of new dependencies on proprietary tools and components. Incorrect markings are often copied to other material, so incorrect markings can "spread" to other projects.

An OSS project must reject any contribution that does not meet the OSS project's chosen license(s). Similarly, an OGOTS project must reject contributions that do not permit OTD development. In particular, an OGOTS project should reject contributions with only "restricted rights" as defined in DFARS 252.227-7014(a)(14) as these do not provide the government and contractors with sufficient rights to reuse the software in arbitrary government circumstances.

References

[Bacon2009] Bacon, Jono. August 2009. The Art of Community: Building the New Age of Participation. ISBN: 978-0-596-15671-8. http://www.artofcommunityonline.org/downloads/jonobacon-theartofcommunity-1ed.pdf

[Fogel2009] Fogel, Karl. 2009. *Producing Open Source Software: How to Run a Successful Free Software Project*. http://producingoss.com/

[Martin2000] Martin, Robert C. 2000. *Design Principles and Design Patterns*. http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf

The U.S. government also has unlimited rights to this document per DFARS 252.227-7013.

Work sponsored by the Assistant Secretary of Defense (Networks & Information Integration) (NII) / DoD Chief Information Officer (CIO) and the Under Secretary of Defense for Acquisition, Technology, and Logistics (AT&L)

Publicly Releasing Open Source Software Developed for the U.S. Government

By Dr. David A. Wheeler

This article summarizes when the U.S. federal government or its contractors may publicly release, as open source software (OSS), software developed with government funds. This article is intended for non-lawyers, to help them understand the basic rules they must follow.

Before going further, a few definitions and warnings are necessary. In this article, the term "government" means the U.S. federal government. "You" means the government organization or contractor who wants to release software to the public as OSS. "Releasing to the public as OSS" means (1) releasing the software source code to the general public (such as through a public website) ***and*** (2) giving its users the freedom to use it (for *any* purpose), study it, modify it, and redistribute it (modified or not)[1]. Note that these freedoms can be given by releasing the software under an OSS license,[2] or by releasing it without any copyright protection. This article is not legal advice, and variations of specific facts can produce different results. Also, note that government contracting is *very* different from commercial practices; do not presume that commercial practices apply.

To determine if you can release to the public some software developed with government funds as OSS, you must answer the following five questions:

1. What contract applies, what are its terms, and what decisions have been made?

First, find the contract and find what terms apply, particularly which data rights clauses apply. Most contracts use one of a small set of standard data rights clauses, but you need to find out *which* clauses apply, and if the contract grants exceptions. If the clause text is different (e.g., older) than the clauses discussed here, or makes an exception, then the contract (if legal) governs. Also, determine what data rights decisions have been made by the contracting officer.

2. Do you have the necessary copyright-related rights?

The following table shows the default copyright-related rights for common circumstances. The first row is a special case, where a federal employee develops the software as part of his or her official duties. Later rows discuss the typical impact of common data rights clauses from the Federal Acquisition Regulation (FAR) or the Department of Defense FAR Supplement (DFARS) (but note the dates):

These are the general rules, but you must examine your specific circumstances to determine exactly what you can do. There are details in the FAR and DFARS clauses not emphasized here, and the contract can change from these defaults to something very different. Some contracts will use different versions of the FAR and DFARS clauses, so check to see if there are any relevant differences. Note that some other agencies (like NASA) have FAR supplements, which are not covered here.

The table above *only* applies to software that was either (1) developed by a government employee as part of his or her official duties or (2) developed by a government contractor (directly or indirectly) as part of a government contract. Such software may include or depend on *other* software, such as commercial software, that does *not* meet these criteria. When a system includes commercial software, the *commercial license* applies to those components, and everyone must comply with their license terms. Commercial software includes any software that is used for at least one non-governmental use and has been sold, leased, or licensed to the general public (per 41 USC §403 and DFARS 252.227-7014(a)(1)), so nearly all publicly-available OSS is commercial software. Commercial software with minor modifications is still considered commercial software.

In many cases the contractor receives copyright. When there are multiple contractors or suppliers (e.g., a lead integrator and subcontractors), the legal arrangements between the

1 This is, in summarized form, the Free Software Definition (http://www.gnu.org/philosophy/free-sw.html) from the Free Software Foundation. A similar definition is in the DoD's "Clarifying Guidance Regarding Open Source Software (OSS)" (http://cio-nii.defense.gov/sites/oss/2009OSS.pdf). A more detailed definition of OSS is the Open Source Definition (http://www.opensource.org/osd.html) from the Open Source Initiative.

2 To release under an OSS license you must have the copyright-related rights (listed in 17 USC §106) to reproduce the work, to prepare derivative works, to distribute copies, and to permit others to perform those actions.

Circumstance	Other Conditions (if any)	Case	Can government release as OSS?	Can contractor release as OSS?
U.S. federal government employee (including military personnel) develops software as part of his/her official duties. This makes it a "Work of the U.S. government."		A	Effectively yes. The software is not subject to copyright protection in the U.S. per 17 USC §105, so if released, anyone in the U.S. can read, use, modify, and redistribute it. The government may apply for copyright outside U.S., but still release the software as OSS.	N/A
FAR 52.227-14 contract clause defaults (December 2007), software first produced in performance of contract.	Government has not granted the contractor the right to assert copyright (default).	B	Yes. The government normally has unlimited rights (essentially the same rights as a copyright holder) per (b)(1). In the FAR source code is software, and software is data, so source code is data.	No. The contractor may request permission to assert copyright.
	Government has granted the contractor the right to assert copyright (e.g., via specific written permission or via clause alternate IV).	C	No. The government does not have sufficient rights, per (c)(1)(iii); it cannot distribute copies to the public. The government should be wary of granting a request to assert copyright, as it permanently loses many rights to data it paid to develop.	Yes. The contractor may assert copyright.
DFARS 252.227-7014 contract clause defaults (June 1995).	Developed exclusively with government funds.	D	Yes. The government has unlimited rights (essentially the same rights as a copyright holder). Per (b)(2)(ii), the 5-year period from mixed funding can be negotiated to a different length of time, and it starts "upon execution of the contract, subcontract, letter contract (or similar contractual instrument), contract modification, or option exercise that required development of the computer software."	Yes. Copyright is held by the contractor/ supplier
	Developed by mixed funding (government partly paid for its development) and (sub)contract execution/mod more than 5 years ago.	E		
	Developed by mixed funding (government partly paid for its development) and (sub)contract execution/mod less than 5 years ago.	F	No. The government does not have sufficient rights. Per (b)(2)(ii), the 5-year period from mixed funding can be negotiated to a different length of time; during this time the government only has "government purpose rights." If software is developed exclusively at private expense, by default the government only has "restricted rights"; the government should be wary of dependencies on such components. The government can negotiate for greater rights per (b)(3) and (b)(4).	
	Developed exclusively at private expense.	G		

Table 1: The default copy-related rights for common circumstances. Continues on next page.

organizations determine *which* contractors/suppliers are legally allowed to assert copyright. Lead contractors do *not* necessarily receive copyrights from their subcontractors and suppliers. Note that the government can receive and hold copyrights transferred to it, per 17 USC §105.

In many cases the government is not the copyright holder but has *unlimited rights* (see rows B, D, E, and I). If the government has unlimited rights, it has essentially the same rights as a copyright holder for purposes of releasing the software as OSS[3]. Thus, it can release the software under any OSS license it chooses, including the GNU General Public License (GPL) and Lesser GPL (LGPL)[4]. When the

3 The Council on Governmental Relations (CAGR)'s "Technical Data and Computer Software: A Guide to Rights and Responsibilities Under Federal Contracts, Grants and Cooperative Agreements" states that "This unlimited license enables the government to act on its own behalf and to authorize others to do the same things that it can do, thus giving the government *essentially* the same rights as the copyright owner."

4 CENDI's "Frequently Asked Questions about Copyright and Computer Software" at http://cendi.gov/publications/09-1FAQ_OpenSourceSoftware_FINAL_110109.pdf question 4.3 says: "an agency may distribute software created by a vendor to all users under an open source licensing scheme if it acquired suf-

Circumstance	Other Conditions (if any)		Case	Can government release as OSS?	Can contractor release as OSS?
DFARS 252.227-7018 contract clause defaults (June 1995): Small Business Innovation Research (SBIR) Program.	Not developed exclusively at private expense.	Less than five years after completion of the project	H	No. The government does not have sufficient rights, per (b)(4)(i).	Yes. The contractor has copyright.
		More than five years after completion of the project and alternate I is not used.	I	Yes. The government has unlimited rights (essentially the same rights as a copyright holder) per (b)(1)(vi). Unfortunately, it is sometimes difficult to determine when the time period has expired.	
		More than five years after completion of the project and alternate I is used.	J	Sometimes. Under alternate I the Government cannot exercise its rights to release if, within certain time limits, the software is published and the contracting officer is notified. This limitation continues as long as it is reasonably available to the public for purchase (after which the government can release it as OSS). See alternate I for details.	
	Developed exclusively at private expense.		K	No. The government does not have sufficient rights, per (b)(2).	
FAR 52.227-17 "Special works" contract clause defaults (December 2007)	Government has not granted the contractor the right to assert copyright (default), and the software was first produced in perform¬ance of the contract.		L	Yes, either through unlimited rights or by holding copyright. By default, the government has unlimited rights in all data delivered under the contract, and in all data first produced in the performance of the contract, per (b)(1)(i). Per (c)(ii), if the contractor has not been granted permission to assert copyright rights, the contracting officer can direct the contractor to assign copyright to the government.	No. Contractor cannot assert copyright rights per (c)(1)(i). The contractor may request permission to assert copyright; if granted see below.
	Government has granted the contractor the right to assert copyright, and the software was first produced in perform¬ance of the contract.		M	No. The government only has the more limited rights listed at the end of (c)(1)(i), and these rights are limited to uses "by or on behalf of the Government."	Yes. Contractor has copyright.
	Software not first produced in the performance of this contract.		N	It depends. Note that a contractor cannot include copyrighted software into a deliverable without written permission of the contracting officer, see (c)(2) for more.	It depends.
DFARS 252.227-7020 "Special works" contract clause defaults (June 1995).	Work first produced, created, or generated and required to be delivered under the contract.		O	Yes. The government receives the copyright, per (c)(2).	No. The government has copyright.
	Other copyrighted works incorporated into a required deliverable (unless written approval granted for an exception).		P	Normally yes. Per (c)(3) and (d), the contractor must normally grant to the government a long list of data rights when incorporating other copyrighted works, and these rights permit OSS release. The contractor may only incorporate software without those rights into a deliverable if the government contracting officer gives written approval, per (d).	Normally yes. The contractor must already have the rights for OSS release to incorporate it, unless given written approval.

government has unlimited rights but is not the copyright holder, there are a few actions it cannot take, e.g., the right to transfer or assign rights, and standing to sue in court over copyright infringement[5]. However, for the purposes here these are technicalities; the key point is that the government can release the software as OSS, under any OSS license it chooses, once it receives unlimited rights.

The government should be extremely wary of receiving less than unlimited rights for software or systems it paid to develop. For example, some contractors will intentionally embed components over which they have exclusive control, and then design the rest of the system to depend on those components. When the government has less than unlimited rights, it risks creating a dependency on a contractor, rendering competition for that system meaningless[6] and in some cases putting military capability at risk.[7] [8]

Some have misunderstood U.S. law and policy as requiring the government to mindlessly accept proposals which give less than unlimited rights for systems developed though government funding. It is true that 10 U.S.C. §2320(a)(2)(F) states that "a contractor or subcontractor (or a prospective contractor or subcontractor) may not be required, as a condition of being responsive to a solicitation or as a condition for the award of a contract, to … sell or otherwise relinquish to the United States any rights in technical data [except in certain cases, and may not be required to] refrain from offering to use, or from using, an item or process to which the contractor is entitled to restrict rights in data"[9]. However, this is not the whole story. "If the Government has properly required certain data or software in a solicitation, it is entitled to certain rights in accordance with the statute and an offer failing to propose at least those rights could be held unacceptable." What is more, the government may (and should) evaluate proposals "on the basis of data rights and giving higher ratings to offerors willing to provide more than the bare minimum rights"[10]

Under many of the FAR (but not DFARS) clauses, if the government agrees to allow contractors to assert copyright, the government *loses* many of its rights, forever, to software that the government paid to develop (see rows B, C, L, and M). This loss of rights can be quite detrimental to the government. What's more, it creates a difficult decision for a contracting officer to make, as the contracting officer must anticipate all possible future uses to make a good decision (something that is difficult in practice). The usual DFARS clause (252.227-7014) avoids this problem; in this clause, typically the government ends up with unlimited rights to software it paid to develop (in some cases after a delay), and the contractor has copyright, enabling *both* parties to take actions such as releasing the software as OSS.

Here are a few notes about specific clauses:

- Under FAR 52.227-14 (rows B and C), the government can grant a contractor the right to assert copyright, at which point the contractor gains rights but the government permanently *loses* rights. Per FAR 27.404-3(a)(2), the government should grant this request *only* "when [that] will enhance appropriate dissemination or use." Government officials should *not* grant this automatically, as doing so dramatically reduces the government's rights to software that the government paid to develop. The government could choose to grant this permission on condition that the software be immediately released to the public under some specific OSS licenses (with the license agreed upon as part of the condition for release). In such a case, public release as OSS would be used as a

ficient rights from the vendor to do so in the software. For example, an "unlimited rights license" acquired under a DFARS procurement-type contract..." Similarly, the "DoD Open Source Software (OSS) FAQ" says that once the government has unlimited rights, it can "use those rights to release that software under a variety of conditions (including an open source software license), because it has the use and modify the software at will, and has the right to authorize others to do so."

5 The government can probably take other measures against someone who does not comply with the license, though. For example, the government may be able to sue for breach of license. Also, an infringer may lose any ability to enforce rights over the resulting work in U.S. court due to the doctrine of unclean hands.

6 Ashton B. Carter, "Memorandum to Acquisition Professionals Subject: Better Buying Power: Mandate for Restoring Affordability and Productivity in Defense Spending" https://dap.dau.mil/policy/Documents/Policy/Carter Memo on Defense Spending 28 Jun 2010.pdf - His first point on providing incentives is to "Avoid directed buys and other substitutes for real competition. Use technical data packages and open systems architectures to support a continuous competitive environment."

7 GAO GAO-06-839 "WEAPONS ACQUISITION: DOD Should Strengthen Policies for Assessing Technical Data Needs to Support Weapon Systems" (July 2006) http://www.gao.gov/new.items/d06839.pdf reported that "The lack of technical data rights has limited the services' flexibility to make changes to sustainment plans that are aimed at achieving cost savings and meeting legislative requirements regarding depot maintenance capabilities... Unless DOD assesses and secures its rights for the use of technical data early in the weapon system acquisition process when it has the greatest leverage to negotiate, DOD may face later challenges in sustaining weapon systems over their life cycle."

8 See, for example, "Fire support's dependence on contractors," Sgt Timothy Caucutt, http://www.mca-marines.org/gazette/article/paying-pirates

9 This U.S. law does not cover software, but the DoD also applies this to software per DFARS 227.7203-1(c) and (d).

10 George O. Winborne, Jr., "Who's Killing the Goose?" American Bar Association Section of Public Contract Law Program Intellectual Property in Government Contracts—What You Didn't Learn in Kindergarten, November 11-12, 2010, Seaport Hotel, Boston, Massachusetts. https://acc.dau.mil/adl/en-US/401584/file/54029/Winborne_ABAPCL_paper_Who's_Killing_the_Goose_For _Release.pdf

method to enhance dissemination and use. Deliverables can include data not first produced in the performance of the contract, per (c)(2), but in this case it is not clear to this author if the government can release software as OSS.

- Under DFARS 252.227-7014 (rows D-G), the contractor normally gets copyright. The government gets the same rights as a copyright holder (via unlimited rights) if (1) the software was developed exclusively with government funding or (2) the funding was mixed and five years have passed after the relevant contract or contract modification that caused its development was signed. The government should beware of situations where the contractor attempts to deliver software that vitally depends on some component that they developed entirely at private expense. Such a dependency can inhibit any future competition for maintenance, as by default the government only has restricted rights to such components.
- Under DFARS 252.227-7018 (rows H-J), the government typically gets unlimited rights to software not exclusively developed at private expense, but only after five years after the project has *completed* (note that this is a different starting time than DFARS 252.227-7014). Amendment I can remove this right as long as the product is "reasonably available to the public for purchase."
- FAR 52.227-17 (rows L-N) is, according to FAR 27.409(e), to be used for software for the "government's internal use" or where "there is a need to limit distribution" or to "obtain indemnities for liabilities." However, purposes change; software originally developed for the "government's internal use" may become software that should be publicly released as OSS. This document simply describes what is allowed, rather than the expectations of the original contract authors.
- DFARS 252.227-7020 (rows O-P, the special works clause) is discussed in DFARS 227.7106. That discussion does not specifically mention software, but the -7020 clause can be used for software. DFARS 227.7106(2) says it can be used for "a work" and is not just limited to "technical data." This clause should be used if the government must own or control copyright. For example, it might be appropriate if the government wishes publicly release OSS and be able to (1) directly enforce copyright in court, and/or (2) provide indemnification.

3. Do you have the necessary other intellectual rights (e.g., patents)?

You need to make sure that you have any other necessary intellectual rights. Most importantly, determine if there are any relevant patents, and if so, what the rights to them are.

Other potential issues are trademark, trade dress, government seals, and trade secrets. Trademark issues, if relevant, can often be easily addressed by simply removing the trademark marking. If the contractor has granted copyright or unlimited rights to the government, then the government already has the rights to release that information to the public and is thus not barred from public release by trade secret law.

4. Do you have permission to release to the public?

In particular, for public release the material must not be restricted by:

- *Classification*. Classified data cannot be legally released to the public. Where this is not obvious, a classification review may be required.
- *Distribution statements*. A government contracting officer may require certain clauses be included in data (including software) to limit its release; contractors must obey these clauses or cause them to be rescinded.
- *Export controls*. The Export Administration Regulations (EAR) are issued by the Department of Commerce), and the International Traffic in Arms Regulations (ITAR) are issued by the Department of State. These prohibit the unlicensed export of specific technologies for reasons of national security or protection of trade. Note that cryptography can invoke export control issues.

Export controls can be particularly confusing, and the penalties for failing to comply with them can be stiff (including large fees and jail time). Thus, here are some basics about export control:

- More information about export control regulations under the EAR are provided by the U.S. Department of Commerce Bureau of Industry and Security (BIS). In particular, see their pages on "Export Control Basics" and "Licensing Guidance." Any item (including software) that is sent from the US to a foreign destination, including to any foreign national, is an export – even if the item originally came from outside the US. Certain U.S. exports/re- exports require a license from BIS. A key is knowing whether the item you are intending to export has a specific Export Control Classification Number (ECCN) as listed in the Commerce Control List (CCL), available on the EAR website. In addition, a license is required for virtually all exports and many re-exports to embargoed destinations and countries designated as supporting terrorist activities.

Continued on page 38

Two Great Reliability Solutions from the RIAC & DACS.

System Reliability Toolkit

RIAC

The RIAC/DACS System Reliability Toolkit provides technical guidance in all aspects of system reliability, allowing the user to understand and implement techniques to ensure that system and product designs exhibit satisfactory hardware, software and human reliability, and to minimize the inherent risks associated with deficiencies in system reliability.

To purchase, please contact:
The Reliability Information Analysis Center
100 Seymour Road
Suite C-101
Utica, NY 13502
1-877-363-RIAC
http://theRIAC.org

The DACS Software Reliability Sourcebook

DACS
Data & Analysis Center for Software

This first edition of the DACS Software Reliability Sourcebook provides a concise resource for information about the practical application of software reliability technology and techniques. The Sourcebook is divided into nine major sections, plus seven supporting appendices.

To purchase, please contact:
The Data & Analysis Center for Software
100 Seymour Road
Suite C-102
Utica, NY 13502
1-800-214-7921
http://thedacs.com

- Similarly, more information about the export control regulations under the ITAR, which implements the Arms Export Control Act (AECA), are provided by the U.S. Department of State Directorate of Defense Trade Controls (DDTC) (http://www.pmddtc.state.gov). In particular, see their page on "Getting Started." The US regulates exports and re-exports of defense items and technologies, so if what you wish to export is covered by the U.S. Munitions List (USML), a license from DDTC is required. You may file a commodity jurisdiction request (CJ) to determine whether an item or service is covered by the U.S. Munitions List (USML) and therefore subject to export controls related to AECA and ITAR.

The Department of Defense (DoD) does ***not*** have authority to grant export control licenses. A contractor may be liable if he or she relies on a DoD official's permission for export control, because in most cases the DoD does not have this authority. Note that even when an export-controlled release of software is granted, it is often contingent on not releasing the source code, making such "releases" useless for open technology development among all parties.

However, ***if the DoD determines that something it has purview over is releasable to the public, it is no longer subject to export control***. This is because 15 C.F.R. 734.3(b)(3) says that "The following items are not subject to the EAR . . . Publicly available technology and software...." Similarly, 22 CFR 125.4 (13) notes that technical data is exempt from ITAR export controls if it is "approved for public release (i.e., unlimited distribution) by the cognizant U.S. government department or agency or Office of Freedom of Information and Security Review." Thus, ***if software is intended to be released to the public, having the cognizant U.S. government department or agency (such as the DoD) approve its public release is often the best way to fully comply with export control regulations***.

5. Do you have the materials (e.g., source code) and are all materials properly marked?

The government and upper-tier contractors should ensure that they receive *all* material, including source code, that they are entitled to. It is all too common to have the right to the source code or related materials, yet not have it and thus be unable to exercise your rights. Source code is necessary for potential OSS release, and it is also necessary to enable competition for future software maintenance bids. Both the government and contractors should make sure that they do not lose the source code, but instead treat it as valuable data (e.g., by creating multiple backup copies in different locations).

Under DFARS 252.227-7014, the definition of "computer software" includes not only "computer programs" but also "source code, source code listings... and related material that would enable the software to be reproduced, recreated, or recompiled." Thus, a delivery of developed software is *supposed* to include source code by default. Also, (b)(1)(i) and (b)(2)(i) state that the government has rights to software (whether it was a deliverable or not) if its funds were used.

Source code should only be accepted if is ready for use. Material should only be considered acceptable as source code if it is the preferred form of the work for making modifications to it. Source code should *not* be accepted if it is just a printout or electronic images of a printout. It should not be accepted unless it is easy to automatically rebuild, e.g., a "make" or similar simple command should be sufficient to recreate an executable. Build documentation should be included with any deliverable, including information on what is required to rebuild it.

It would be best if the source code also included the historical record (e.g., a complete record of each change, who made it, and when), in an electronic form adequate for transfer to another configuration management system. Ideally, the government should have sufficient access to the software engineering environment of the contractor, so that the government could monitor changes as they are made.

Ensure that the source code and other materials are marked appropriately. Companies may include restrictive markings on materials, and if those markings are inappropriate, then the markings need to be fixed. Government contract clauses include processes for fixing incorrect markings; follow them. Government and upper-tier contractors need to promptly challenge improperly marked materials due to time limits. For example, contracts using DFARS 227.7203-13 include, in item (d)(3)(i), a challenge time limit of three years after either the final payment or the delivery of software, whichever is later. Also, improper markings tend to be copied into other materials; fixing markings early greatly reduces the effort to fix them later.

Who has authority?

Unfortunately, it is not always obvious *who* in government or the various contractors can make these decisions. It would be best if the government and contractors could clarify roles,

policies, and procedures. In the meantime, the following may be helpful:

- As noted above, when there are multiple contractors or suppliers, the legal arrangements between the organizations determine *which* contractors/suppliers are legally allowed to assert copyright. Lead contractors do *not* necessarily receive copyrights from their subcontractors and suppliers. By U.S. law (17 USC §201), "Copyright... vests initially in the author or authors of the work... In the case of a work made for hire, the employer or other person for whom the work was prepared is considered the author [and holds the copyright] unless the parties have expressly agreed otherwise in a written instrument signed by them."
- The 2009 DoD OSS memo *does* clarify who in the DoD can determine when it should release software as OSS, and under what conditions. It says that "Software items, including code fixes and enhancements, developed for the Government should be released to the public (such as under an open source license) when all of the following conditions are met:
    1. The project manager, program manager, or other comparable official determines that it is in the Government's interest to do so, such as through the expectation of future enhancements by others.
    2. The Government has the rights to reproduce and release the item, and to authorize others to do so. For example, the Government has public release rights when the software is developed by Government personnel, when the Government receives "unlimited rights" in software developed by a contractor at Government expense, or when pre-existing OSS is modified by or for the Government.
    3. The public release of the item is not restricted by other law or regulation, such as the Export Administration Regulations or the International Traffic in Arms Regulation, and the item qualifies for Distribution Statement A, per DoD Directive 5230.24 (reference (i))."
- Some organizations do not have a review process for software source code but *do* have a process for reviewing documents. In these cases, it may be appropriate to submit the source code to the document review process. This is especially relevant for classification review.

Final notes

If the government and relevant contractors intend to release software as OSS, it's best if that is explicitly stated ahead of time. For example, OSS could be identified as the planned software maintenance philosophy per DFARS 227.7203-2(b)(1). However, since many contracts do not discuss releasing software as OSS, it's important to understand the default rules for commonly-encountered cases.

If software is released to the public as OSS and it becomes "customarily used by the general public or by nongovernmental entities for purposes other than governmental purposes," then that software becomes *commercial software*. This is by both law (41 USC §403) and regulation (e.g., DFARS 252.227-7014(a)(1)). It does not matter if the software was originally developed with government funds, or not. Thus, releasing software as OSS can be a commercialization approach.

The U.S. government and its contractors have released many programs as OSS. I hope that this material helps you understand how you can release software as OSS in a manner consistent with laws, regulations, and contracts.

The publication of this paper does not indicate endorsement by the Department of Defense or IDA, nor should the contents be construed as reflecting the official positions of those organizations.

The DACS Gold Practice Initiative:

- Promotes effective selection/use of software acquisition & development practices
- Defines essential activities/benefits of each practice
- Considers the environment in which each practice is used
- Addresses the timeliness of practice benefits
- Recognizes interrelationships between practices that influence success or failure
- Contains quantitative and qualitative information
- A continually evolving resource for the DoD, Government, Industry and Academia
- Free to use/free to join

Learn More About the DACS Gold Practice Initiative:
http://www.goldpractices.com

Current Gold Practices:

- Acquisition Process Improvement
- Architecture-First Approach
- Assess Reuse Risks and Costs
- Binary Quality Gates at the Inch-Pebble Level
- Capture Artifacts in Rigorous, Model-Based Notation
- Commercial Specifications and Standards/Open Systems
- Defect Tracking Against Quality Targets
- Develop and Maintain a Life Cycle Business Case
- Ensure Interoperability
- Formal Inspections
- Formal Risk Management
- Goal-Question-Metric Approach
- Integrated product and Process Development
- Metrics-Based Scheduling
- Model-Based Testing
- Plan for Technology Insertion
- Requirements Management
- Requirements Trade-Off/Negotiations
- Statistical Process Control
- Track Earned Value

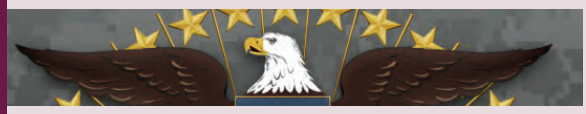
THE DACS GOLD PRACTICE INITIATIVE

DACS

The Data & Analysis Center for Software

100 Seymour Road
Utica, NY 13502
http://www.thedacs.com

Additional OSS Resources

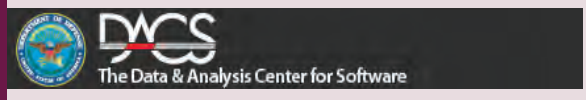


- Mil-OSS connects and empowers an active community of civilian and military open source software and hardware developers across the United States: **http://www.mil-oss.org/**

MIL-OSS Google Group**: http://groups.google.com/group/mil-oss?hl=en**

- A coalition organized to serve as a centralized advocate, to encourage broader U.S. Federal Government support of and participation in Open Source projects and Technologies: **http://opensourceforamerica.org/**

- GOSCON is the Government Open Source Conference, an annual event produced by Oregon State University's Open Source Lab: **http://goscon.org/**



- The DACS topic page contains information relative to the development, use, licensing and promotion of open source software including operating systems, browsers and applications: **https://www.thedacs.com/databases/url/key/4878**

- Frequently Asked Questions about Copyright and Computer Software - Issues Affecting the U.S. Government with Special Emphasis on Open Source Software (CENDI/09-1) (Updated October 2010):

http://www.cendi.gov/publications/09-1FAQ_OpenSourceSoftware_FINAL_110109.pdf

- This memorandum provides clarifying guidance on the use of OSS and supersedes the previous DoD CIO memorandum dated May 28,2003 (reference (a)):

http://cio-nii.defense.gov/sites/oss/

- *Producing Open Source Software* is a book about the human side of open source development. It describes how successful projects operate, the expectations of users and developers, and the culture of free software. The book is released under an open copyright: it is available in bookstores and from the publisher (O'Reilly Media), or you can browse or download it here: **http://producingoss.com/**

STN Article Submission Policy

The STN is a theme-based quarterly journal. In the past DACS has typically solicited specific authors to participate in developing each theme, but we recognize that it is not possible for us to know about all the experts, programs, and work being done and we may be missing some important contributions. In 2009 DACS adopted a policy of accepting articles submitted by the software professional community for consideration.

DACS will review articles and assist candidate authors in creating the final draft if the article is selected for publication. Note that DACS does not pay for articles published. Note also that submittal of an article constitutes a transfer of ownership from the author to DACS with DACS holding the copyright.

Although the STN is theme-based, we do not limit the content of the issue strictly to that theme. If you submit an article that DACS deems to be worthy of sharing with the community, DACS will find a way to get it published. However, we cannot guarantee publication within a fixed time frame in that situation. Consult the theme selection page and the Author Guidelines located on the STN web site (see **https://www.softwaretechnews.com/**) for further details.

To submit material (or ask questions) contact **news-editor@thedacs.com**

Recent themes include:

- Earned Value
- Software Testing
- Project Management
- Model Driven Development
- Software Quality and Reliability
- Cyber Security

ABOUT THE SOFTWARE TECH NEWS

STN EDITORIAL BOARD

Distribution Statement:
Unclassified and Unlimited

DACS
100 Seymour Road
Utica, NY 13502-1348
Phone: 800-214-7921
Fax: 315-732-3261
E-mail: news-editor@thedacs.com
URL: http://www.thedacs.com/

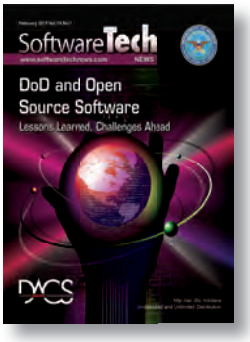
ADVERTISEMENTS

The Software Tech News is now accepting advertisements for future newsletters. In addition to being seen by the thousands of people who subscribe to a paper copy, an electronic version is available at the DACS website, exposing your product, organization, or service to hundreds of thousands of additional eyes every month.

For rates and layout information contact: news-editor@thedacs.com

COVER DESIGN

Shelley Howard
Graphic Designer
Quanterion Solutions, DACS



ARTICLE REPRODUCTION

Images and information presented in these articles may be reproduced as long as the following message is noted:

"This article was originally published in the Software Tech News, Vol. 14, No. 1 February 2011."
Requests for copies of the referenced newsletter may be submitted to the following address:

Data & Analysis Center for Software
100 Seymour Road
Utica, NY 13502-1348

Phone: 800-214-7921
Fax: 315-732-3261
E-mail: news-editor@thedacs.com
An archive of past newsletters is available at **www.SoftwareTechNews.com.** In addition to this print message, we ask that you notify DACS regarding any document that references any article appearing in the *Software Tech News*.

ABOUT THIS PUBLICATION

The Software Tech News is published quarterly by the Data & Analysis Center for Software (DACS). The DACS is a DoD sponsored Information Analysis Center (IAC), administratively managed by the Defense Technical Information Center (DTIC). The DACS is technically managed by Air Force Research Laboratory, Rome, NY and operated by Quanterion Solutions Incorporated.

Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the DACS. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the DACS, and shall not be used for advertising or product endorsement purposes.

Data & Analysis Center for Software
100 Seymour Road
Suite C-102
Utica, NY 13502

Return Service Requested

STN 14-1 February 2011: DoD and Open Source Software

IN THIS ISSUE